HANDOUTS | MATERIAL
HIGHLIGHTED AND PROVIDED BY :

VU TALK

CLICK TO JOIN OUR WHATSAPP GROUP : click me

VU TALK YouTube CHANNEL : click me

For Any Help Contact Me On Whatsapp : click me

Follow VU TALK on Whatsapp Channel : click me

HUMAN COMPUTER INTERACTION

(CS408)

TABLE OF CONTENTS:

LECTURE 1. 8

INTRODUCTION TO HUMAN COMPUTER INTERACTION – PART I 8

LEARNING GOALS 8
1.1 RIDDLES FOR THE INFORMATION AGE 10
1.2 ROLE OF HCI 14

LECTURE 2. 15

INTRODUCTION TO HUMAN-COMPUTER INTERACTION – PART II 15

LEARNING GOALS 15
2.1 DEFINITION OF HCI 15
2.2 REASONS OF NON-BRIGHT ASPECTS 15
2.3 HUMAN VERSES COMPUTER 18
2.4 SOFTWARE APARTHEID 19

LECTURE 3. 22

INTRODUCTION TO HUMAN-COMPUTER INTERACTION – PART III 22

LEARNING GOALS 22
3.1 AN INDUSTRY IN DENIAL 23
3.2 TECHNO-RAGE 24
3.3 SUCCESS CRITERIA IN THE NEW ECONOMY 26
3.4 COMPUTER + INFORMATION 28
REFERENCES 29

LECTURE 4. 30

GOALS & EVOLUTION OF HUMAN COMPUTER INTERACTION 30

LEARNING GOALS 30
4.1 GOALS OF HCI 30
4.2 EVOLUTION OF HCI 34

LECTURE 5. 39

DISCIPLINE OF HUMAN COMPUTER INTERACTION 39

LEARNING GOALS 39
5.1 QUALITY 39
5.2 INTERDISCIPLINARY NATURE OF HCI 41
REFERENCE: 45

LECTURE 6. 46

COGNITIVE FRAMEWORKS 46

LEARNING GOALS 46

6.1 INTRODUCTION 46
6.2 MODES OF COGNITION 48
6.3 HUMAN PROCESSOR MODEL 49
6.4 GOMS 50
6.5 RECENT DEVELOPMENT IN COGNITIVE PSYCHOLOGY 51
6.6 EXTERNAL COGNITION 52
6.7 DISTRIBUTED COGNITION 53

LECTURE 7. 54

HUMAN INPUT-OUTPUT CHANNELS – PART I 54

LEARNING GOALS 54
7.1 INPUT OUTPUT CHANNELS 54

© Copyright Virtual University of Pakistan

7.2 VISION 55
7.3 VISUAL PERCEPTION 57

LECTURE 8. 62

HUMAN INPUT-OUTPUT CHANNELS PART II 62

LEARNING GOALS 62
8.1 COLOR THEORY 62
8.2 STEREOPSIS 66
8.3 READING 69
8.4 HEARING 69
8.5 TOUCH 71
8.6 MOVEMENT 72

LECTURE 9. 76

COGNITIVE PROCESS - PART I 76

LEARNING GOALS 76
9.1 ATTENTION 76
9.2 MEMORY 79
9.3 REVISED MEMORY MODEL 84

LECTURE 10. 85

COGNITIVE PROCESSES - PART II 85

LEARNING GOALS 85
10.1 LEARNING 85
10.2 READING, SPEAKING AND LISTENING 87
10.3 PROBLEM SOLVING, PLANNING, REASONING AND DECISION-MAKING 88

LECTURE 11. 93

THE PSYCHOLOGY OF ACTIONS 93

LEARNING GOALS 93
11.1 MENTAL MODEL 93
11.2 ERRORS 100

LECTURE 12. 101

DESIGN PRINCIPLES 101

LEARNING GOALS 101
12.1 DESIGN PRINCIPLES 103

LECTURE 13. 109

THE COMPUTER 109

LEARNING GOALS 109
13.1 INPUT DEVICES 109
13.2 TEXT ENTRY DEVICES 110
13.3 POSITIONING, POINTING AND DRAWING 115
13.4 DISPLAY DEVICES 118
13.5 TOUCH, FEEL AND SMELL 120
13.6 PHYSICAL CONTROLS 120
13.7 ENVIRONMENT AND BIO SENSING 120

LECTURE 14. 121

INTERACTION 121

LEARNING GOALS 121
14.1 THE TERMS OF INTERACTION 121
14.2 DONALD NORMAN'S MODEL 121
14.3 THE INTERACTION FRAMEWORK 122
14.4 FRAMEWORKS AND HCI 123

© Copyright Virtual University of Pakistan

14.5 INTERACTION STYLES 127
LECTURE 15. 130
INTERACTION PARADIGMS 130
LEARNING GOALS 130
15.1 THE WIMP INTERFACES 130
15.2 INTERACTION PARADIGMS 134
LECTURE 16. 141
HCI PROCESS AND MODELS 141
LEARNING GOALS 141
LECTURE 17. 148
HCI PROCESS AND METHODOLOGIES 148
LEARNING GOALS 148
17.1 LIFECYCLE MODELS 149
17.2 LIFECYCLE MODELS IN HCI 152
LECTURE 18. 157
GOAL-DIRECTED DESIGN METHODOLOGIES 157
LEARNING GOALS 157
18.1 GOAL-DIRECTED DESIGN MODEL 157
18.2 A PROCESS OVERVIEW 158
18.3 TYPES OF USERS 162
LECTURE 19. 166
USER RESEARCH PART-I 166
LEARNING GOALS 166
19.1 TYPES OF QUALITATIVE RESEARCH 167
LECTURE 20. 172
USER RESEARCH PART-II 172
LEARNING GOALS 172
20.1 USER-CENTERED APPROACH 172
20.2 ETHNOGRAPHY FRAMEWORK 174
20.3 PREPARING FOR ETHNOGRAPHIC INTERVIEWS 178
20.4 PUTTING A PLAN TOGETHER 180
LECTURE 21. 181
USER RESEARCH PART-III 181
LECTURE 22. 185
USER MODELING 185
LEARNING GOALS 185
22.1 WHY MODEL? 185
22.2 PERSONAS 186
22.3 GOALS 191
22.4 TYPES OF GOALS 191
22.5 CONSTRUCTING PERSONAS 194
LECTURE 23. 198
REQUIREMENTS 198
LEARNING GOALS 198
23.1 NARRATIVE AS A DESIGN TOOL 198
23.2 ENVISIONING SOLUTIONS WITH PERSONA-BASED DESIGN 200

© Copyright Virtual University of Pakistan

LECTURE 24. 205

FRAMEWORK AND REFINEMENTS 205

LEARNING GOALS 205
24.1 DEFINING THE INTERACTION FRAMEWORK 205
24.2 PROTOTYPING 209

LECTURE 25. 212

DESIGN SYNTHESIS 212

LEARNING GOALS 212
25.1 INTERACTION DESIGN PRINCIPLES 212
25.2 INTERACTION DESIGN PATTERNS 216
25.3 INTERACTION DESIGN IMPERATIVES 218

LECTURE 26. 220

BEHAVIOR & FORM PART I 220

LEARNING GOALS 220
26.1 SOFTWARE POSTURE 220
26.2 POSTURES FOR THE DESKTOP 220

LECTURE 27. 229

BEHAVIOR & FORM PART II 229

LEARNING GOALS 229
27.1 POSTURES FOR THE WEB 229
27.2 WEB PORTALS 230
27.3 POSTURES FOR OTHER PLATFORMS 231
27.4 FLOW AND TRANSPARENCY 233
27.5 ORCHESTRATION 236

LECTURE 28. 245

BEHAVIOR & FORM PART III 245

LEARNING GOALS 245
28.1 ELIMINATING EXCISE 245
28.2 NAVIGATION AND INFLECTION 248

LECTURE 29. 256

EVALUATION – PART I 256

LEARNING GOALS 256
29.1 EVALUATION PARADIGMS AND TECHNIQUES 258

LECTURE 30. 264

EVALUATION – PART II 264

LEARNING GOALS 264
30.1 DECIDE: A FRAMEWORK TO GUIDE EVALUATION 264

LECTURE 31. 270

EVALUATION – PART VII 270

LEARNING GOALS 270

LECTURE 32. 279

EVALUATION IV 279

LEARNING GOALS 279
32.1 SCENE FROM A MALL 279
32.2 WEB NAVIGATION 281

© Copyright Virtual University of Pakistan

LECTURE 33. 294

EVALUATION V 294

LEARNING GOALS 294
33.1 TRY THE TRUNK TEST 296

LECTURE 34. 300

EVALUATION – PART VI 300

LEARNING GOALS 300

LECTURE 35. 304

EVALUATION – PART VII 304

LEARNING GOALS 304
35.1 THE RELATIONSHIP BETWEEN EVALUATION AND USABILITY? 304

LECTURE 36. 310

BEHAVIOR & FORM – PART IV 310

LEARNING GOALS 310
36.1 UNDERSTANDING UNDO 310
36.2 TYPES AND VARIANTS OF 312
36.3 INCREMENTAL AND PROCEDURAL ACTIONS 312
36.4 SINGLE AND MULTIPLE UNDO 312
36.5 OTHER MODELS FOR UNDO-LIKE BEHAVIOR 316
36.6 RETHINKING FILES AND SAVE 319
36.7 ARCHIVING 321
36.8 IMPLEMENTATION MODEL VERSUS MENTAL MODEL 322
36.9 DISPENSING WITH THE IMPLEMENTATION MODEL OF THE FILE SYSTEM 323
36.10 DESIGNING A UNIFIED FILE PRESENTATION MODEL 323

LECTURE 37. 325

BEHAVIOR & FORM - PART V 325

LEARNING GOALS 325
37.1 UNIFIED DOCUMENT MANAGEMENT 325
37.2 CREATING A MILESTONE COPY OF THE DOCUMENT 328
37.3 ARE DISKS AND FILES SYSTEMS A FEATURE? 329
37.4 TIME FOR CHANGE 330
37.5 MAKING SOFTWARE CONSIDERATE 330
37.6 CONSIDERATE SOFTWARE IS POSSIBLE 336
37.7 MAKING SOFTWARE SMARTS: 336
37.8 PUTTING THE IDLE CYCLES TO WORK 336
37.9 GIVING SOFTWARE A MEMORY 337
37.10 TASK COHERENCE 338
37.11 ACTIONS TO REMEMBER 339
37.12 APPLYING MEMORY TO YOUR APPLICATIONS 340
37.13 MEMORY MAKES A DIFFERENCE 342

LECTURE 38. 343

BEHAVIOR & FORM – PART VI 343

LEARNING GOALS 343
38.1 DESIGNING LOOK AND FEEL 343
38.2 PRINCIPLES OF VISUAL INTERFACE DESIGN 345

LECTURE 39. 348

BEHAVIOR & FORM – PART VII 348

LEARNING GOALS 348
39.1 PROVIDE VISUAL STRUCTURE AND FLOW AT EACH LEVEL OFORGANIZATION 348

© Copyright Virtual University of Pakistan

39.2 PRINCIPLES OF VISUAL INFORMATION DESIGN 354
39.3 USE OF TEXT AND COLOR IN VISUAL INTERFACES 356
39.4 CONSISTENCY AND STANDARDS 358

LECTURE 40. 361

OBSERVING USER 361

LEARNING GOALS 361
40.1 WHAT AND WHEN TO OBSERVE 361
40.2 HOW TO OBSERVE 361
40.3 DATA COLLECTION 365
40.4 INDIRECT OBSERVATION: TRACKING USERS' ACTIVITIES 366
40.5 ANALYZING, INTERPRETING, AND PRESENTING THE DATA 367

LECTURE 41. 370

ASKING USERS 370

LEARNING GOALS 370
41.1 INTRODUCTION 370
41.2 ASKING USERS: INTERVIEWS 370
41.3 ASKING USERS: QUESTIONNAIRES 374
41.4 ASKING EXPERTS: WALKTHROUGHS 379

LECTURE 42. 381

COMMUNICATING USERS 381

LEARNING GOALS 381
42.1 ELIMINATING ERRORS 381
42.2 POSITIVE FEEDBACK 385
42.3 NOTIFYING AND CONFIRMING 387
42.4 ALERTS AND CONFIRMATIONS 387
42.5 ELIMINATING CONFIRMATIONS 390
42.6 REPLACING DIALOGS: RICH MODELESS FEEDBACK 391
42.7 RICH VISUAL MODELESS FEEDBACK 391

LECTURE 43. 393

INFORMATION RETRIEVAL 393

LEARNING GOALS 393
43.1 AUDIBLE FEEDBACK 393
43.2 OTHER COMMUNICATION WITH USERS 395
43.3 IMPROVING DATA RETRIEVAL 401

LECTURE 44. 406

EMERGING PARADIGMS 406

LEARNING GOALS 406
44.1 ACCESSIBILITY 408

LECTURE 45. 413

CONCLUSION 413

LEARNING GOALS 413
45.1 WEARABLE COMPUTING 414
45.2 TANGIBLE BITS 416
45.3 ATTENTIVE ENVIRONMENTS 418

© Copyright Virtual University of Pakistan

Lecture 1.

Introduction to Human Computer Interaction – Part I

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

Answer what is the significance of Human Computer Interaction (HCI)

Discuss and argue about why Human computer Interaction (HCI) is important with reference to the way in which technology has developed during past forty years

Describe a formal definition of HCI.

At the end of this lecture you will be told about the course contents. This will be a brief overview of the topics that we will discuss in this course and the structure of the course.

Run for your lives---invasion has begun---the computers are invading.

Now it is twenty first century and during the past thirty years technology has advanced to such an extent that almost everyone come in contact with computers in one way or another. Look around yourself how many things are there which have some kind of computer embedded in them? Think about a minute about what you use in a typical day; ATM, cell phone, VCR, remote control, ticketing machine, digital personal organizers, calculator, watch, photocopier, toaster, bank, air conditioner, broadcasting, satellite, microwave, medical equipment, factories, companies….the list is endless. Computers are everywhere. We are surrounded by computers. Now they are part of our everyday life.

Traditional notion of computers is no more. Unlike in the early days of computing, when only highly skilled technical people used computers, nowadays the range of knowledge and experience of different users is very broad. Computers are no more just on your table. Now computer has become a tool of everyday use. They are everywhere, at everyplace and in everything. They are penetrating in every aspect of our life. They are taking our lives.

When computers first appeared on the commercial scene in the 1950s, they were extremely difficult to use, cumbersome and at times unpredictable. There were a number of reasons for this;

They were very large and expensive machines, so that by comparison human labor (that is, 'people time') was an inexpensive resource.
They were used only by technical specialists – scientists and engineers – who were familiar with the intricacies of off-line programming using punch cards.

© Copyright Virtual University of Pakistan

Little was known about how to make them easier to use.
None of these conditions holds today: computers have become much less expensive, users come from every walk of life, and we understand a great deal more about how to fit the machines to people's needs and their work.

Dramatic decrease in the cost of computing resources have resulted from new technological advances, the most significant being the development of the silicon chip. The ability not only to miniaturize circuits but also to pack large number of them on to tiny, individual chips paved the way for his development of powerful computers with large storage capacity. In less than thirty years computers changed from being huge machines housed in large, air-conditioned rooms to much smaller machines, including some that can easily be carried around by children. Computers have also become more reliable and today's machines do not suffer from overheating like their ancestors. Computing has entered a new era and is becoming ubiquitous.

The development of the first personal computers in the 1970s was a major landmark because these machines provided interactive computing power for individual users at low cost. Consequently, instead of just a handful of highly experienced programmers being the only users, people from all walks of life – commerce, farming, education, retailing, defense, manufacturing and entertainment – began using computer systems. Computers are becoming increasingly POWERFUL.

Computers are performing more and more TASKS. These changes in technology have opened up a wide range of new possibilities for the way in which computers can be used. The sheer costliness and time required to run programs on the early machines dictated the kinds of commercial application in which computers could be used. Business such as banking and accounting, with large-scale record keeping activities, were the first to take up computing technology. Companies that were involved in activities with 'fast' cycles, such as transaction processing for airlines and retailing, could not make use of these machines. They were not sufficiently fast or responsive, but this is not a problem with modern computers.

Computers have also found a place in many private homes. In fact, such has been their pervasiveness that now just about everyone, young or old, able or disabled, skilled or unskilled, is using or is directly affected by computers in one way or another. Machines are leaking into every aspect of lives. So now the concept of life, without computer is same as concept of life without electricity, and obviously it is hard to live without light as well as with computer!

Run for your lives---invasion has begun---the computers are invading.
As computers are penetrating in our daily life, it has some results. The bright side of this invasion is:

Computers are enabling new discoveries
Leading to efficiencies
Making our life easy and convenient

On the not so bright side the result is:

© Copyright Virtual University of Pakistan

Computers are annoying us
They are infuriating us
They even kill a few of us.

In turn, we will be tempted to kill our computers, but we won't dare because we are already utterly, irreversibly dependent on these hopeful monsters that make modern life possible. So we will have to think about them. We will have to think how we can make them better. We need to fundamentally rethink how human and machines interact. And rethink the relationship in deep and novel ways, for the fault for our burgeoning problems lies not with our machines, but with us.

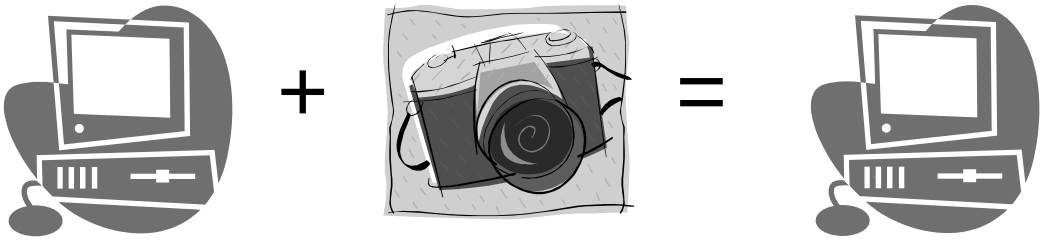
1.1 Riddles for the Information Age

What do you get when you cross a computer with an Airplane?



In December 1995, American Airlines Flight 965 departed from Miami on a regularly scheduled trip to Cali, Columbia. On the landing approach, the pilot of the 757 needed to select the next radio navigation fix, named "ROZO". He entered an "R" into his navigation computer. The computer returned a list of nearby navigation fixes starting with "R" and the pilot selected the first of these, whose latitude and longitude appeared to be correct. Unfortunately, instead of "ROZO", the pilot selected "ROMEO", 132 miles to the northeast. The jet was southbound descending into a valley that runs north-south, and any lateral deviation was dangerous. Following indications on the flight computer, the pilots began an easterly turn and slammed into a granite peak at 10,000 feet. One hundred and fifty two passengers and all eight crewmembers aboard perished. Four passengers survived with serious injuries.

What do you get when you cross a computer with a Camera?



Here is a riddle for the information age: what do you get when you cross a computer with a camera? Answer: A computer! Thirty years ago, a 35mm Pentax Model H, had a small

© Copyright Virtual University of Pakistan

battery in it that powered the light meter. Like a wristwatch battery, I merely swapped in a new one every couple of years. Fifteen years ago, an electronic camera, a 35mm Canon T70, used two AA batteries to power its rather simple exposure computer and its automatic film drive. It had a simple On/Off switch, so that the batteries wouldn't wear down needlessly.

Five years ago, a first-generation digital camera, had a similar On/Off switch, but this time it had the smarts of a rudimentary computer inside it. So if I forgot to turn it off, it automatically shut down after one minute of inactivity.

One year ago, second-generation digital camera, a Panasonic PalmCam, had an even smarter computer chip inside it. It was so smart that its On/Off switch had evolved into an Off/Rec/Play switch. It now had modes: it had to put into Rec mode to take pictures and Play mode to view them on its small video display.

The newest camera, a Nikon CoolPix 900, is a third-generation digital camera and the smartest yet. In fact, it has a full-blown computer that displays a Windows-like hourglass while it "boots up". Like some mutant fish with extra heads, its On/Off switch has now grown to have four settings: Off/ARec/MRec/Play. "ARec" means "automatic record" and "MRec" means "manual record." as far as I can figure out how to turn it on without a lengthy explanation.

The new camera is very power-hungry, and its engineers thoughtfully provided it with a sophisticated computer program that manages the consumption of battery power. A typical scenario goes like this: I turn the evil off/etc. switch to "MRec," wait about seven long seconds for the camera to boot up, then point it at my subject. I aim the camera and zoom in to properly frame the image. Just as I'm about to press the shutter button, the camera suddenly realizes that simultaneously running the zoom, charging the flash, and energizing the display has caused it to run out of power. In self-defense, it suspends its ability to actually take pictures. But I don't know that because I'm liking through the viewfinder, waving my arms and saying "Smile" and pressing the shutter button. The computer detects the button press, but it simply cannot obey. In a misguided effort to help out, the power management program instantly takes over and makes an executive decision: shed load. It shuts down the power-greedy LCD video display. I look at the camera quizzically, wondering why it didn't take the picture, shrug my shoulders, and let my arm holding the camera drop to my side. But as soon as the LCD is turned off, there is more battery power available for other systems. The power management program senses this increase and realizes that it now has enough electricity to take pictures. It now returns control to the camera program, which is waiting patiently to process the command it received when I pressed the shutter button, and it takes a nicely auto-focused, well-exposed, high-resolution digital picture of my kneecap.

That old mechanical Pentax had manual focusing, manual exposure, and manual shutter-speed, yet it was far less frustrating to use than the fully computerized modern Nikon CoolPix 900, which has automatic focusing, exposure, and shutter-speed. Camera may still take pictures, but it behaves like a computer instead of a camera.

© Copyright Virtual University of Pakistan

What do you get when you cross a computer with an alarm clock?



A computer! I just purchased an expensive new clock-radio for my bedroom, a JVC FS-2000. It has a very sophisticated computer brain, and offers high fidelity, digital sound, and lots of features. It wakes me up at a preset time by playing a compact disc, and it has the delicacy and intelligence to slowly fade up the volume when it begins to play at six o'clock in the morning. This feature is really pleasant and quite unique, and it compensates for the fact that I want to hurl the infuriating machine out the window.

It's very hard to tell when the alarm is armed, so it occasionally fails to wake me up on a Monday and rousts me out of bed early on a Saturday. Sure, it has an indicator to show the alarm is set, but that doesn't mean it's useful. The clock has a sophisticated alphanumeric liquid crystal display (LCD) that displays all of its many functions. The presence of a small symbol in the upper left corner of the LCD indicates the alarm is armed, but in a dimly lit bedroom the clock symbol visible, but the backlight comes on when the CD or radio is explicitly turned on. There's a gotcha, however, as the alarm simply won't ever sound while the CD is explicitly left on, regardless of the setting of the alarm. It is this paradoxical operation that frequently catches me unaware.

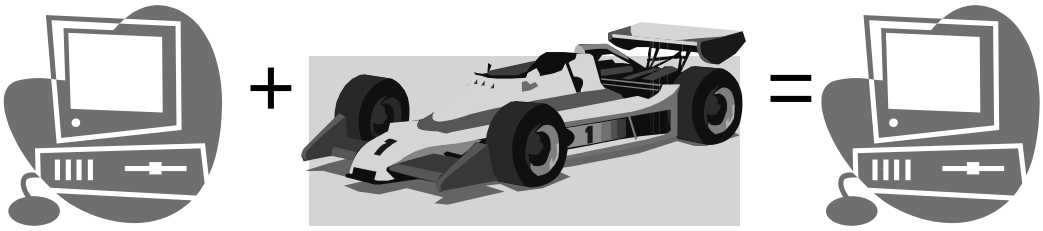
It is simple to disarm the alarm: Simply press the "Alarm" button once, and the clock symbol disappears from the display. However to arm it, I must press the "Alarm" button exactly five times. The first time I press it, the display shows me the time of the alarm. On press tow, it shows the time when it will turn the sound off. On press three, it shows me whether it will play the radio or the CD. On press four, it shows me the preset volume. On press five, it returns to the normal view, but with the alarm now armed. But with just one additional press, it disarms the alarm. Sleepy, in a dark bedroom, it is quite difficult to perform this little digital ballet correctly. The alarm clock may still wake me up, but it behaves like a computer.

By contrast, my old non-computerized alarm clock woke me up with a sudden, unholy buzzing. When it was armed, a single red light glowed. When it was not armed, the red light was dark. I didn't like this old alarm clock for many reasons, but at least I could tell when it was going to wake me up.

Because it is far cheaper for manufacturers to use computers to control the internal functioning of devices than it is to use older, mechanical methods, it is economically inevitable that computers will insinuate themselves into every product and service in our lives. This means that the behavior of all of our products will be the same as most obnoxious computers, unless we try some thing different.

© Copyright Virtual University of Pakistan

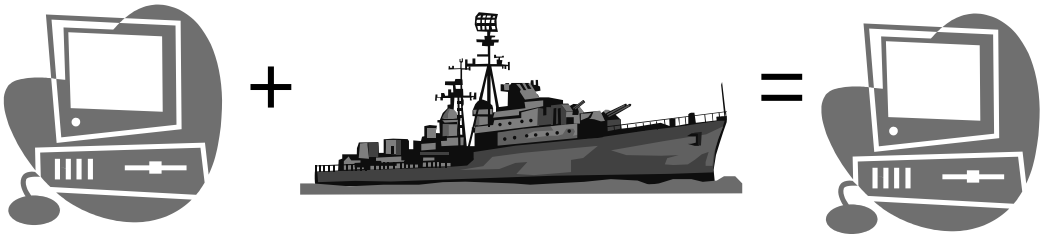
What do you get when you cross a computer with a car?



A computer! Porsche's beautiful new high-tech spots car, the Boxster, has seven computers in it to help manage its complex systems. One of them is dedicated to managing the engine. It has special procedures built into it to deal with abnormal situations. Unfortunately, these sometimes backfire. In some early models, if the fuel level in the gas tank got very low---only a gallon or so remaining---the centrifugal force of a sharp turn could cause the fuel to collect in the side of the tank, allowing air to enter the fuel lines. The computer sensed this as a dramatic change in the in coming fuel mixture, and interpreted it as a catastrophic failure of the injection system. To prevent damage, the computer would shut down the ignition and stop the car. Also to prevent damage, the computer would not let the driver restart the engine until the car had been towed to a shock and serviced

When owners of early Boxsters first discovered this problem, the only solution Porsche could devise was to tell them to open the engine compartment and disconnect the battery for at least five minutes, giving the computer time to forget all knowledge of the hiccup. The sports car may still speed down those too-lane black top roads, but now, in those turns, it behaves like a computer.

What do you get when you cross a computer with a warship?



In September of 1997, while conducting fleet maneuvers in the Atlantic, the USS Yorktown, one of the Navy's new Aegis guided-missile cruisers, stopped dead in the water. A Navy technician, while calibrating an on-board fuel valve, entered a zero into one of the shipboard management computers, a Pentium Pro running Windows NT. The program attempted to divide another number by that zero---a mathematically undefined operation---which resulted in a complete crash of the entire shipboard control system. Without the computers, the engine halted and the ship sat wallowing in the swells for two

© Copyright Virtual University of Pakistan

hours and fifty-five minutes until it could be towed into port. Good thing it wasn't in a war zone.

What do you get when you cross a computer with a warship? Admiral Nimitz is rolling in his grave! Despite this setback, the Navy is committed to computerizing all of its ships because of the manpower cost savings, and so deflect criticism of this plan, it has blamed the "incident" on human error. Because the software creation process is out of control, the high-tech industry must either bring its process to heel or it will continue to put the blame on ordinary users while ever-bigger machines sit dead in the water

So here you saw the result of integrating computers in our lives. As I said early, computers will annoy us, infuriate us, and even kill a few of us. In turn, we will be tempted to kill our computers, but we won't dare because we are already utterly, irreversibly dependent on these hopeful monsters that make modern life possible. So we will have to think about them. We will have to think how we can make them better. We need to fundamentally rethink how human and machines interact. And rethink the relationship in deep and novel ways, for the fault for our burgeoning problems lies not with our machines, but with us.

1.2 Role of HCI

Here comes the role of HCI. Human designed the interfaces we hate; human continue to use dysfunctional machines even as the awkward interfaces strain their eyes, ache their backs, and ruin their wrist tendons. HCI plays a role to bridge up the gape between the interfaces of machines and human understanding that we have seen in the previous examples.

Definition of HCI

"Human-Computer Interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them"
-ACM/IEEE

© Copyright Virtual University of Pakistan

Lecture 2.

Introduction to Human-Computer Interaction – Part II

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Describe the significance of HCI, particularly adverse impact of computer technology on humans and reasons for these adverse effects
- Describe the nature of humans and computers
- Understand the Paradox of the computing phenomena
- Differentiate between focus of SE and HCI

2.1 Definition of HCI

"Human-Computer Interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them"
-ACM/IEEE

2.2 Reasons of non-bright Aspects

Airplane + Computer

In last lecture we were discussing the incident of airplane. Today we will look at the reason of such a fatal incident.



The National Transportation Safety Board investigated, and ---as usual---declared the problem human error. The navigational aid the pilots were following was valid but not for the landing procedure at Cali. In the literal definition of the phrase, this was indeed human error, because the pilot selected the wrong fix. However, in the larger picture, it was not the pilot's fault at all.

© Copyright Virtual University of Pakistan

The front panel of the airplane's navigation computer showed the currently selected navigation fix and a course deviation indicator. When the plane is on course, the needle is centered, but the needle gives no indication whatsoever about the correctness of the selected radio beacon. The gauge looks pretty much the same just before landing as it does just before crashing. The computer told the pilot he was tracking precisely to the beacon he had selected. Unfortunately, it neglected to tell him the beacon the selected was a fatal choice.

The flight computer on Flight 965 could easily have told the pilots that ROMEO was not an appropriate fix for their approach to Cali. Even a simple hint that it was "unusual" or "unfamiliar" could have saved the airplane. Instead, it seemed as though the computer was utterly unconcerned with the actual flight and its passengers. It cared only about its own internal computations

Joke in Computer Industry

There is a widely told joke in the computer industry that goes like this: A man is flying in a small airplane and is lost in the clouds. He descends until he spots an office building and yells to a man in an open window, "Where am I?" The man replies, "You are in an airplane about 100 feet above the ground." The pilot immediately turns to the proper course, spots the airport and lands. His astonished passenger asks how the pilot figured out which way to go. The pilot replies, "The answer the man gave me was completely correct and factual, yet it was no help whatsoever, so I knew immediately he was a software engineer who worked for Microsoft and I know where Microsoft's building is in relation to the airport."

When seen in the light of the tragedy of Flight 965, the humor of the joke is macabre, yet professionals in the digital world tell it gleefully and frequently because it highlights a fundamental truth about computers:

They may tell us facts but they don't inform us.

They may guide us with precision but they don't guide us where we want to go. The flight computer on Flight 965 could easily have told the pilots that ROMEO was not an appropriate fix for their approach to Cali. Even a simple hint that it was "unusual" or "unfamiliar" could have saved the airplane. Instead, it seemed as though the computer was utterly unconcerned with the actual flight and its passengers. It cared only about its own internal computations

Communication can be precise and exacting while still being tragically wrong. This happens all too frequently when we communicate with computers, and computers are invading every aspect of our modern lives. From the planes we fly to just about every consumer product and service, computers are ubiquitous, and so is their characteristically poor way of communicating and behaving.[1]

© Copyright Virtual University of Pakistan

I-Drive Car Device







It takes automotive computer power to a whole new level. Computer systems provide the car with BMW's most powerful engine, a silky smooth ride and what is supposed to be the simplest in-dash control system available. But what is created for the sake of simplicity can often time creates the most confusion.

Many controls are operated with a single large, multifunction knob located in the console between the front seats. The control consists of a combination rotary and push button for selecting functions. Confirmation of the selected mode is displayed on a dash-mounted screen.

Users can change functions -- from communications to climate control, navigation or entertainment -- by pushing the console knob forward or back, or side-to-side. By twisting the knob, they can scroll through menus. And by clicking a button located in the middle of the knob, they can select functions.

"iDrive" takes into account the fact that comfort, communication and driver assistance functions are only rarely adjusted while driving. The operating unit in the center console gives the driver direct access to many other driving functions and information and communication options. Several hundred functions can be controlled with this device.

A computer-type monitor is positioned directly within the driver's line of vision to the road ahead. The large monitor in the center of the dashboard displays all the information the driver needs, apart from the speedometer and tachometer, which are conventional analog instruments.

> The driver slides the dial to choose between multiple control menus displayed on an in-dash LCD screen. The driver rotates the dial to move through lists and pushes the dial axially to select a list item.

After reading that I didn't feel like I had any sort of idea what 'axially' meant, but I suppose this video helps. What concerns me about this is the interaction with this little device requires the driver, hurtling down the road, to look at a screen. They say there is force feedback that indicates the menu, but that's only half the equation, because there are things in the menus. So, I'm guessing the driver needs to memorize the menus, which are sure to be short, so think about the mental modeling here.

To really keep your eyes on the road, you have to be able to do everything by feel and pattern. Is this easier than hot-cold air sliders, vent selection buttons and radio dials?

© Copyright Virtual University of Pakistan

It takes 15 minutes to change a Radio Channel. The fundamental flaw: you absolutely have to take your eyes off the road to change settings. Result is constant Calls to Help Desk

Feature Shock

Every digital device has more features than its manual counterpart, but manual devices easier to use. Hi-tech companies add more features to improve product. Product becomes complicated

Bad process can't improve product

Computer + Bank

A computer! Whenever I withdraw cash from an automatic teller machine (ATM), I encounter the same sullen and difficult behavior so universal with computers. If I make the slightest mistake, it rejects the entire transaction and kicks me out of the process. I have to pull my card out, reinsert it, reenter my PIN code, and then re-assert my request. Typically, it wasn't my mistake, either, but the ATM computer finesses me into a misstep. It always asks me whether I want to withdraw money from my checking, saving, or money market account, even though I have only checking account. Subsequently, I always forget which type it is, and the question confuses me. About once a month I inadvertently select "savings", and the infernal machine summarily boots me out of the entire transaction to start over the beginning. To reject "savings", the machine has to know that I don't have a saving account, yet it still offers it to me as a choice. The only difference between me selecting "saving" and the pilot of Flight 965 selecting "ROMEO" is the magnitude of the penalty.

The ATM has rules that must be followed, and I am quite willing to follow them, but it is unreasonably computer-like to fail to inform me of them, giving me contradictory indications, and then summarily punish me for innocently transgressing them. This behavior---so typical of computers---is not intrinsic to them. Actually nothing is intrinsic to computers: they merely act on behalf of their software, the program. And programs are as malleable as human speech. A person can speak rudely of politely, helpfully or sullenly. It is as simple for a computer to behave with respect and courtesy as it is for a human to speak that way. All it takes is for someone to describe how. Unfortunately, programmers aren't very good at teaching that to computers.

In order to solve some of these problems, here comes the relatively new and emerging field of Human Computer Interaction (HCI).[1]

2.3 Human verses Computer

Human species

Human beings are the most interesting and fascinating specie on planet. They are the most complex living being on the earth. It has very much diversity in its nature. It is intelligent in its deeds. Human beings think and decide according to their own will. Yes, they are free in nature. They like freedom. They think on a problem dynamically and they can find many solutions that may not exist before. They can invent. They are not only rational but they also have emotions. They also think emotionally. They act emotionally.

© Copyright Virtual University of Pakistan

And fortunately or unfortunately they make mistakes. They make mistakes which some time become fatal for them and some time they become blessing for them.

Computer species

On contrast, computers are the invention of human being. They are also complex but they are also pretty dumb. It can also think but it can't think on its own will, it thinks how it has been directed to think. No doubt its speed is marvelous. It does not tire. It is emotionless. It has no feelings, no desires. It works how it has been commanded to work. And they do not make mistakes.

Before penetration of computers in our daily life, human beings were performing their tasks at their on responsibility. In a business domain human beings were dealing and interacting with each other's. For example a store manager was dealing with all the workers performing their different duties in the store. Some one was registering the new arrivals of products, some one was numbering the products and many more…and store manager has to interact with all these human beings. If some one was a salesperson, he used to interact with different clients and used to deal with them according to their mood and desire. He could judge their mood with their tone, their attitude and with their body language. He could provide answers relevant to their questions.

But now in this age of information technology we are expecting computers to mimic human behavior e.g. ECommerce systems, now there is no need for a salesperson. Web sites are behaving as a salesperson or as a shopping mal. That is now; a dumb, unintelligent and inanimate object will perform the complex task which was performed by some human being.

2.4 Software Apartheid

Apartheid

Racial segregation; specifically: a policy of segregation and political and economic discrimination against non-European groups in the Republic of South Africa. [Definition of apartheid]

Software Apartheid

Institutionalizing obnoxious behavior and obscure interactions of software-based products. [Definition of software apartheid]

Programmers generally work in high-tech environments, surrounded by their technical peers in enclaves like Silicon Valley. Software engineers constantly encounter their peers when they shop, dine out, take their kids to school and relax, while their contact with frustrated computer users is limited. What's more, the occasional unfocused gripes of the users are offset by the frequent enthusiasm of the knowledgeable elite. We forget how far removed our peers and we are from the frustration and inability of the rest of the country (not to mention the world) to use interactive tools.

We industry insiders toss around the term "computer literacy", assuming that in order to use computers; people must acquire some fundamental level of training. We see this as a

© Copyright Virtual University of Pakistan

simple demand that is not hard and is only right and proper. We imagine that it is not much to ask of users that they grasp the rudiments of how the machines work in order to enjoy their benefits. But it is too much to ask. Having a computer literate customer base makes the development process much easier—of their can be no doubt—but it hampers the growth and success of the industry and of society. Apologists counter with the argument that you must have training and a license to drive a car, but they overlook the fact that a mistake with software generally does not. If cars were not so deadly, people would train themselves to derive the same way they learn excel.

It has another, more insidious effect. It creates a demarcation line between the haves and have-nots in society. If you must master a computer in order to succeed in America's job Market beyond a burger-flipper's carriers, then the difficulty of mastering interactive systems forces many people into menial jobs rather than allowing them to matriculate into more productive, respected and better-paying jobs.

Users should not have to acquire computer literacy to use computer for common, rudimentary task in everyday life. Users should not have to possess a digital sensitivity to work their VCR, microwave oven, or to get e-mail. What's more, should not have to acquire computer literacy to use computer for enterprise applications, where the user is already trained in the application domain. An accountant for example, who is trained in the general principles of accounting, should not have to become computer literate to use a computer in her accounting practice. Her domain knowledge should be enough to see her through.

As our economy shifts more and more onto information bases, we are inadvertently creating a divided society. The upper class is composed of those who have mastered the nuances of differentiating between "RAM" and "Hard Disk". The lower class is that who treat the difference inconsequential. The irony is that the difference really is inconsequential to any one except a few hard-core engineers. Yet virtually all-contemporary software forces its users to confront a file system, where your success fully dependent on knowing the difference between RAM and disk.

Thus the term "computer literacy" becomes a euphemism for social and economic apartheid. Computer literacy is a key phrase that brutally bifurcates our society.

But about those people who are not inclined to pander to technocrats and who can not or will not become computer literate? These people, many by choice, but most by circumstances, are falling behind in the information revolution. Many high-tech companies, for example, would not even consider for employment any applicant who does not have an e-mail address. I'm sure that there are many otherwise qualified candidates out there who cannot get the hired because they are not yet wired. Despite the claims of the Apologists, using e-mail effectively is difficult and involves a significant level of computer literacy. Therefore, it artificially segregates the work force. It is the model equivalent of the banking technique of "red lining". In this illegal procedure, all houses in a given neighborhood are declared unacceptable as controller for a housing loan. Although the red lines on the map are ostensibly drawn around economic contours, they tend to follow racial lines all too closely bankers protest that they are not racists, but the effect is the same.

© Copyright Virtual University of Pakistan

When programmers speak of "computer literacy", they are drawing red lines around ethnic groups, too, yet few have pointed this out. It is too hard to see what is really happening because the issue is obscured by technical mythology. It is easy to see---regardless of how true---that a banker can make a loan on one house as easily as on another. However, it is not easy to see that a programmer can make interactive products easy enough for people from lower socio-economic backgrounds to use.

"Acceptable levels of quality for software engineers are far lower than are those for traditional engineering disciplines"

"Software-based products not INHERENTLY hard to use Wrong process is used to develop them" [1]

Software Engineering and HCI

There is a basic fundamental difference between the approaches taken by software engineers and human-computer interaction specialists. Human-computer interface specialists are user-centered and software engineers are system-centered.

Software engineering methodologies are good at modeling certain aspects of the problem domain. Formal methods have been developed to represent data, architectural, and procedural aspects of a software system. Software engineering approaches deal with managerial and financial issues well. Software engineering methodologies are useful for specifying and building the functional aspects of a software system.

Human-computer interfaces emphasize developing a deep understanding of user characteristics and a clear awareness of the tasks a user must perform. HCI specialists test design ideas on real users and use formal evaluation techniques to replace intuition in guiding design. This constant reality check improves the final product.

References

[1] The Inmates are running the asylum by Alan Cooper.
[2] Human Computer Interaction by Jenny Preece.

© Copyright Virtual University of Pakistan

Lecture 3.

Introduction to Human-Computer Interaction – Part III

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Discuss the effect of bad tools
- Discuss and argue about why Human computer Interaction (HCI) is important with reference to the way in which technology has developed during past forty years
- Describe requirement of new economy era.

Effect of Bad Tools

Not only are computers taking over the cockpit of jet airliners, they are taking over the passenger cabin, too, behaving in that same obstinate, perverse way that is so easy to recognize and so hard to use. Modern jet planes have in-flight entertainment (IFE) systems that deliver movies and music to airline passengers. These IFEs are merely computers connected with local area network, just like in your office. Advanced IFE systems are generally installed only on larger airplanes flying transoceanic routes.

One airline's IFE was so frustrating for the flight attendants to use that many of them were bidding to fly shorter, local routes to avoid having to learn and use the difficult systems. This is remarkable considering that the time-honored airline route-bidding process is based on seniority, and that those same long-distance routes have always been considered the most desirable plums because of their lengthy layovers in exotic locales like Singapore or Paris. For flight attendants to bid for unglamorous, unromantic yo-yo flights from Denver-to-Dallas or LA-to-San Francisco just to avoid the IFE indicated a serious morale problem. Any airline that inflicted bad tools on its most prized employee—the ones who spent the most time with the customer---was making a foolish decision and was profligately discarding money, customer loyalty, and staff loyalty.

The computer-IFE of another large airline was even worse. The airline had created an in-flight entertainment system that linked movie delivery with the cash collection function. In a sealed et airplane flying at 37,000 feet, cash collection procedures had typically been quite laissez-faire; after all, nobody was going to sneak out the back door. Flight attendants delivered goods and services when it was convenient and collected cash in only a very loosely coupled fashion. This kept them from running unnecessarily up and down the narrow aisles. Sure there were occasional errors, but never more than a few dollars were involved, and the system was quite human and forgiving; everyone was happy and the work was not oppressive.

© Copyright Virtual University of Pakistan

With cash-collection connected to content delivery by computer, the flight attendant had to first get the cash from the passenger, then walk all the way to the head-end of the cabin, where the attendant's console was, enter an attendant password, then perform a cash register-like transaction. Only when that transaction was completed could the passenger actually view a movie or listen to music. This inane product design forced the flight attendants to walk up and down those narrow aisles hundreds of extra times during a typical trip. Out of sheer frustration, the flight attendants would trip the circuit breaker on the in-flight entertainment system at the beginning of each long flight, shortly after departure. They would then blandly announce to the passengers that, sorry, the system was broken and there would be no movie on this flight.

The airline had spent millions of dollars constructing a system so obnoxious that its users deliberately turned it off to avoid interacting with it. The thousands of bored passengers were merely innocent victims. And this happened on long, overseas trips typically packed with much-sought-after frequent flyers. I cannot put a dollar figure

On the expense this caused the airline, but I can say with conviction that it was catastrophically expensive.

The software inside the IFEs worked with flawless precision, but was a resounding failure because it misbehaved with its human keepers.

3.1 An Industry in Denial

We are a world awash in high-tech tools. Computers dominate the workplace and our homes, and vehicles are filling up with silicon-powered gadgets. All of these computerized devices are wildly sophisticated and powerful, but every one of them is dauntingly difficult and confusing to use.

The high-tech industry is in denial of a simple fact that every person with a cell phone or a word processor can clearly see: our computerized tools are hard to use. The technologists who create software and high-tech gadgets are satisfied with their efforts. The software engineers who create them have tried as hard as they can to make them easy to use and they have made some minor progress. They believe that their products are as easy to use as it is technically possible to make them. As engineers, their belief is in technology, and they have faith that only some new technology, like voice recognition or artificial intelligence, will improve the user's experience.

Ironically, the thing that will likely make the least improvement in the ease of use of software-based products is new technology. There is little difference technically between a complicated, confusing program and a simple, fun, and powerful product. The problem is one of culture, training, and attitude of the people who make them, more than it is one of chips and programming languages. We are deficient in our development process, not in out development tools.

The high-tech industry has inadvertently put programmers and engineers in charge, so their hard-to-use engineering culture dominates. Despite appearances, business executives are simply not the ones in control of the high-tech industry. It is the engineers who are running the show. In our rush to accept the many benefits of the silicon chip, we have abdicated our responsibilities. We have let the inmates run the asylum.

© Copyright Virtual University of Pakistan

When the inmates run the asylum, it is hard for them to see clearly the nature of the problems that bedevil them. When you look in the mirror, it is all too easy to single out your best features and overlook the warts. When the creators of software-based products examine their handiwork, they see how rich the product is in features and functions. They ignore how excruciatingly difficult it is to use, how many mind-numbing hours it takes to learn, or how it diminishes and degrades the people who must use it in their everyday lives.

3.2 **Techno-Rage**

An article in a recent issue of the Wall Street Journal described an anonymous video clip circulated widely by email that showed as "…Mustachioed Everyman in a short sleeved shirt hunched over a computer terminal, looking puzzled. Suddenly, he strikes the side of his monitor in frustration. As a curious co-worker peers over his cubicle, the man slams the keyboard into the monitor, knocking it to the floor. Rising from his chair, he goes after the fallen monitor with a final ferocious kick." The article went on to say that reaction to the clip had been "intense" and that it had apparently tapped into a powerful undercurrent of techno-rage".

It's ironic that one needs to be moderately computer savvy to even send or view this video clip. While the man in the video may well be an actor, he touches a widespread, sympathetic chord in out business world. The frustration that difficult and unpleasant software-based products are bringing to our lives is rising rapidly.

Joke email circulates on private lists about "Computer Tourette's." This is a play on the disorder known as Tourette's syndrome, where some sufferers engage in uncontrollable bouts of swearing. The joke is that you can walk down the halls of most modern office buildings and hear otherwise-normal people sitting in front of their monitors, jaws clenched, swearing repeatedly in a rictus of tense fury. Who knows what triggered such an outburst: a misplaced file, an inaccessible image, or a frustrating interaction. Or maybe the program just blandly erased the user's only copy of a 500-page manuscripts because he responded with a "Yes" to a confirmation dialog box, assuming that it had asked him if he wanted to "save your changes?" when it actually asked him if he wanted to "discard your work?"

Novatech survey

One in every four computers has been physically attacked by its owner, according to a survey.

The survey, conducted by British PC manufacturer Novatech, was intended to take a lighthearted look at embarrassing experiences -- those little technical bloopers that happen even to ubergeeks, like forwarding a personal love letter to an entire office mailing list.

But instead, a much darker story emerged in the 4,200 replies. Innocent computers are being beaten on a regular basis, often by technically challenged users who decide to take their frustrations out on their helpless machines.

© Copyright Virtual University of Pakistan

"We decided to do some research into people's relationships with their computers and we were amazed by some of the results," Novatech managing director David Furby said. "As computers become more and more a part of our daily lives, we obviously share more experiences with them."

Many technical support people from the United States, Canada and parts of Europe have sobering stories of brutalized computers being brought in for repair by sad -- or in some cases, smug -- owners who had smacked, kicked or otherwise deliberately injured their machines.

"The incidences of willful neglect have always been high," said David Tepper, owner of the Village Computer Shop in New York. "We've always had to deal with computers damaged by people who dumped their refreshing beverage on the computer's keyboard, or got tangled up in the cords and bringing (sic) the computer crashing down off their desk."

"But there have also always been a small –- but significant -– number of machines that were obviously intentionally damaged."

"Hopefully as technology improves and computers become ever more user- friendly, these attacks will become less frequent," Furby said.

Computer rage

There is a technology-based scourge afoot…maybe. It’s not a virus; it’s not a denial of service attack; it’s computer rage, and according to the latest reports, it is out to destroy the physical health, the emotional stability, and if left unchallenged, the economic strength of whatever population it strikes.

Security software vendor Symantec, working with Britain’s National Opinion Poll, recently found that when confronted with technical problems, more than 40 percent of British users surveyed have sworn at, kicked, or otherwise abused their computers, monitors, and the most victimized of all computer components, their keyboards.

In similar surveys conducted last year, Marlborough, Mass-based Concord Communications discovered that 83 percent of 150 U.S. respondents witnessed such attacks, and the international market research firm Mori found that 40 percent of 1250 British workers had watched as their colleagues leveled verbal and physical abuse at their computers.

Stress related to computer rage, the Symantec study claims, has resulted in a loss of productivity for most respondents.

Robert Edelmann, clinical psychologist and author of Interpersonal Conflicts at Work, is worried. “Frustration with IT should be taken seriously as a modern malaise,” he says. “It is affecting both our work and our home lives to the extent that computer rage is now much more prolific than road rage.”

Computers increasingly commonplace in offices

© Copyright Virtual University of Pakistan

As the reliance on computers in the workplace continues to grow, people in the UK are resorting to violence when their PCs break down, say researchers. When faced with technical problems, most people shouted at colleagues, hit the PC or even threw parts of the computers. The most frustrating hitch was when people lost their work after their computer crashed or froze.

The problems seem to be widespread with more than a quarter of those working with computers experience problems with their PC on a weekly basis.

"Over half of all working days lost to sickness in the UK are related to workplace stress," said Fiona Dennis, a stress management trainer with Priory Healthcare. "Being heavily reliant on IT to speed up our lives means that performance is hampered greatly when it fails, causing an over-reaction and stress."

70% swear at PCs

The study by the National Opinion Poll and the software company Symantec, found that nearly half of all computer users had become angry at some time. Almost a third of people had physically attacked a computer, 67% experienced frustration, exasperation and anger and more than 70% swore at their machines.

Technology rage is the latest rage to emerge in Britain and follows road rage, trolley rage and air rage. There was a dramatic rise in air rage incidents last year, with 174 people detained at Heathrow and Gatwick alone. In 1998 the number of air rage arrests for the whole country was 98.

3.3 Success Criteria in the New Economy

The successful professional for the twenty-first century is either a business savvy technologist or a technology-savvy businessperson.

The technology-savvy businessperson knows that his success is dependent on the quality of the information available to him and the sophistication with which he uses it. The business-savvy technologist, on the other hand, is an entrepreneurial engineer or scientist trained for technology, but possessing a knee business sense and an awareness of the power of information. Both of these new archetypes are coming to dominate contemporary business.

You can divide all businesspeople into two categories: those who will master high technology and those who will soon be going out of business. No longer can executive delegate information processing to specialists. Business is information processing. You differentiate yourself today with the quality of your information-handling systems, not your manufacturing systems. If you manufacture anything, chances are it has a microchip in it. If you offer a service, odds are that offer it with computerized tools. Attempting to identify businesses that depend on high technology is as futile as trying to identify businesses that depend on the telephone. The high-tech revolution has invaded every business, and digital information is the beating heart of your workday.

It has been said, “To err is human; to really screw up, you need a computer.”

© Copyright Virtual University of Pakistan

Inefficient mechanical systems can waste couple cents on every widget you build, but you can lose your entire company to bad information processes. The leverage that software-based products---and the engineers that build them---have on your company is enormous.

Sadly, our digital tools are extremely hard to learn, use, and understand, and they often cause us to fall short of our goals. This wastes money, time, and opportunity. As a business-savvy technologist/ technology-savvy businessperson, you produce software-based products or consume them---probably both. Having better, easier-to-learn, easier-to-use high-tech products is in your personal and professional best interest. Better products don't take longer to create, nor do they cost more to build. The irony is that they don't have to be difficult, but are so only because our process for making them is old-fashioned and needs fixing. Only long-standing traditions rooted in misconceptions keep us from having better products in today.

Consider a scenario: a website is developed of ecommerce system. The site is aesthetically very beautiful, technically it has no flaw and it has wonderful animated content on it. But if user is unable to find its desired information about the products or even he is unable to find the product out of thousands of products, so what of it's use. It is useless from the business point of view.

Here are some facts and figures:

Users can only find information 42% of the time
– Jared Spool

62% of web shoppers give up looking for the item they want to buy online
– Zona Research
–

50% of the potential sales from a site are lost because people cannot find the item they are looking for
– Forrester Research

40% of the users who do not return to a site do so because their first visit resulted in a negative experience
– Forrester Research

80% of software lifecycle costs occur after the product is released, in the maintenance phase - of that work, 80 % is due to unmet or unforeseen user requirements; only 20 % is due to bugs or reliability problems.

- IEEE Software

Around 63% of software projects exceed their cost estimates. The top four reasons for this are:
– Frequent requests for changes from users
– Overlooked tasks
– Users' lack of understanding of their own requirements
– Insufficient user-analyst communication and understanding

© Copyright Virtual University of Pakistan

- Communications of the ACM

BOO.com, a $204m startup fails
– BBC News

Poor commercial web sites will kill 80% of Fortune 500 companies within a decade
- Jakob Nielsen

So all above given facts reveals that the product with the bad user experience deserve to die!

The serious financial implications of today's digital products should not in any ways be underestimated.

The table given below depicts two scenarios of potential of sales from an e-commerce web site. In scenario A, users can easily find items they are looking for, so 0% sales are lost, so the actual revenue is $100 million. In scenario B, users cannot easily find the items they are looking for, therefore, the actual revenue is $50 million, thus causing a loss of $50 million.

	Scenario A	Scenario B
Revenue Potential	$100m	$100m
User Experience	Good	Bad
Sales Lost	0%	50%
Revenue Lost	$0m	$50m
Actual Revenue	**$100m**	**$50m**

3.4 Computer + Information

What do you get when you cross a computer with information?

In 2000, the Olympic Games were held in Sydney. Before the Olympic games could begin, a lawsuit was filed against the Olympic Committee. The case was called Bruce Lindsay Maguire vs Sydney Organizing Committee for the Olympics Games (SOCOG). On 7 June 1999 the complainant, who is blind, complained to the

© Copyright Virtual University of Pakistan

Commission that he was unlawfully discriminated against by the respondent in three respects:

- the failure to provide Braille copies of the information required to place orders for Olympic Games tickets;
- the failure to provide Braille copies of the Olympic Games souvenir programmed; and
- The failure to provide a web site which was accessible to the complainant.

It was alleged that the SOCOG was in breach the *Disability Discrimination Act* 1992 by failing to make accessible to him key parts of its web site

According to the law of many European and western countries, organizations with a web site must ensure that their web site is (within certain limits) accessible by disabled persons. However, this was not the case in the matter of the official Olympic Games web site. Could this have been avoided? Certainly: by applying a few very simple techniques, the developers of the web site could have made it accessible to people with vision-impairment. But as is usually the case, this was not done.

Result: the complainant won the case and was awarded a sum of money in damages. This was very embarrassing for both the SOCOG and also the company that developed the web site.

References

1. http://www.independentliving.org/docs5/sydney-olympics-blind-accessibility-decision.html
2. http://comment.cio.com/soundoff/061400.html

© Copyright Virtual University of Pakistan

Lecture 4.

Goals & Evolution of Human Computer Interaction

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Describe the goals of HCI
- Define Usability goals
- Define User Experience goals
- Discuss the History and Evolution of HCI

Definition of HCI

"Human-Computer Interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them"
-ACM/IEEE

4.1 Goals of HCI

The term Human Computer Interaction (HCI) was adopted in the mid-1980s as a means of describing this new field of study. This term acknowledged that the focus of interest was broader than just the design of the interface and was concerned with all those aspects that relate to the interaction between users and computers.

The goals of HCI are to produce usable and safe systems, as well as functional systems. These goals can be summarized as 'to develop or improve the safety, utility, effectiveness, efficiency and usability of systems that include computers' (Interacting with computers, 1989). In this context the term 'system' derives from systems theory and it refers not just to the hardware and software but to the entire environment---be it organization of people at work at, home or engaged in leisure pursuits---that uses or is affected by the computer technology in question. Utility refers to the functionality of a system or, in other words, the things it can do. Improving effectiveness and efficiency are self-evident and ubiquitous objectives. The promotion of safety in relation to computer systems is of paramount importance in the design of safety-critical systems. Usability, a key concept in HCI, is concerned with making systems easy to learn and easy to use. Poorly designed computer system can be extremely annoying to users, as you can understand from above described incidents. [2]

Part of the process of understanding user's needs, with respect to designing an interactive system to support them, is to be clear about your primary objective. Is it to design a very efficient system that will allow users to be highly productive to their work, or is to design a system that will be challenging and motivating so that it supports effective learning, or is it some thing else? We call these talk-level concerns usability goals and user experience

© Copyright Virtual University of Pakistan

goals. The two differ in terms of how they are operational zed, i.e., how they can be met and through what means. Usability goals are concerned with meeting specific usability criteria (e.g., efficiency) and user experience goals are largely concern with explicating the quality of the user experience (e.g., to be aesthetically pleasing).

Usability goals

To recap, usability in generally regarded as ensuring that interactive products are easy to learn, effective to use, and enjoyable from user perspective.

It involves optimizing the interactions people have with interactive product to enable them to carry out their activities at work, school, and in their everyday life. More specifically, usability is broken down into the following goals:

- Effective to use (effectiveness)
- Efficient to use (efficiency)
- Safe to use(safety)
- Have good utility (utility)
- Easy to learn (learnability)
- Easy to remember how to use (memorability)

For each goal, we describe it in more detail.

Effectiveness

It is a very general goal and refers to how good a system at doing what it is suppose to do. [1]

Efficiency

It refers to the way a system supports users in carrying out their tasks. [1]

Safety

It involves protecting the users from dangerous conditions and undesirable situations. In relation to the first ergonomics aspect, it refers to the external conditions where people work. For example, where there are hazardous conditions---like x-rays machines or chemical plants---operators should be able to interact with and control computer-based system remotely. The second aspect refers to helping any kind of user in any kind of situation avoid the danger of carrying out unwanted action accidentally. It also refers to the perceived fears users might have of the consequences of making errors and how this effects their behavior to make computer-based system safer in this sense involves:

- Preventing the user from making serious error by reducing the risk of wrong keys/buttons being mistakenly activated (an example is not placing the quit or delete-file command right next to the save command on a menu.) and
- Providing users with various means of recovery should they make errors. Save interactive systems should engender confidence and allow the users the opportunity to explore the interface to carry out new operations.

Other safety mechanism include undo facilities and confirmatory dialog boxes that give users another chance to consider their intentions (a well-known used in email application

© Copyright Virtual University of Pakistan

is the appearance of a dialog box after the user has highlighted the messages to be deleted, saying: “are you sure you want to delete all these messages?”)

Utility

It refers to the extent to which the system provides the right kind of functionality so that user can do what they need or want to do. An example of a system with high utility is an accounting software package providing a powerful computational tool that accountants can use to work out tax returns. An example of a system with low utility is a software drawing tool that does not allow users to draw free hand but forces them to use a mouse to create their drawings, using only polygon shapes. [1]

Learnability

It refers to how easy a system is to learn to use. It is well known that people do not like spending a long time learning how to use a system. They want to get started straight away and become competent at caring out tasks without to much effort. This is especially so far interactive products intended for everyday use (for example interactive TV, email) and those used only infrequently (for example, video conferencing) to certain extent, people are prepared to spend longer learning more complex system that provide a wider range of functionality (for example web authoring tools, word processors) in these situations, CD ROM and online tutorials can help by providing interactive step by step material with hands-on exercises. However many people find these tedious and often difficult to relate to the tasks they want to accomplish. A key concern is determining how much time users are prepared to spend learning a system. There seems little point in developing a range of functionality if the majority of users are unable or not prepared to spend time learning how to use it. [1]

Memorability

It refers to how easy a system is to remember how to use, once learned. This is especially important for interactive systems that are used infrequently. If users haven’t used a system or an operation for a few months or longer, they should be able to remember or at least rapidly be reminded how to use it. Users shouldn’t have to keep relearning how to carry out tasks. Unfortunately, this tends to happen when the operation required to be learning are obscure, illogical, or poorly sequenced. Users need to be helped to remember how to do tasks. There are many ways of designing the interaction to support this. For example, users can be helped to remember the sequence of operations at different stages of a task through meaningful icons, command names, and menu options. Also, structuring options and icons so they are placed in relevant categories of options (for example, placing all the drawing tools in the same place on the screen) can help the user remember where to look to find a particular tool at a given stage of a task. [1]

“Don’t Make me THINK, is the key to a usable product”

User experience goals

The realization that new technologies are offering increasing opportunity for supporting people in their everyday lives has led researchers and practitioners to consider further goals. The emergence of technologies (for example, virtual reality, the web, mobile computing) in diversity of application areas (e.g., entertainment, education, home, public areas) has brought about a much wider set of concerns. As well as focusing primarily on

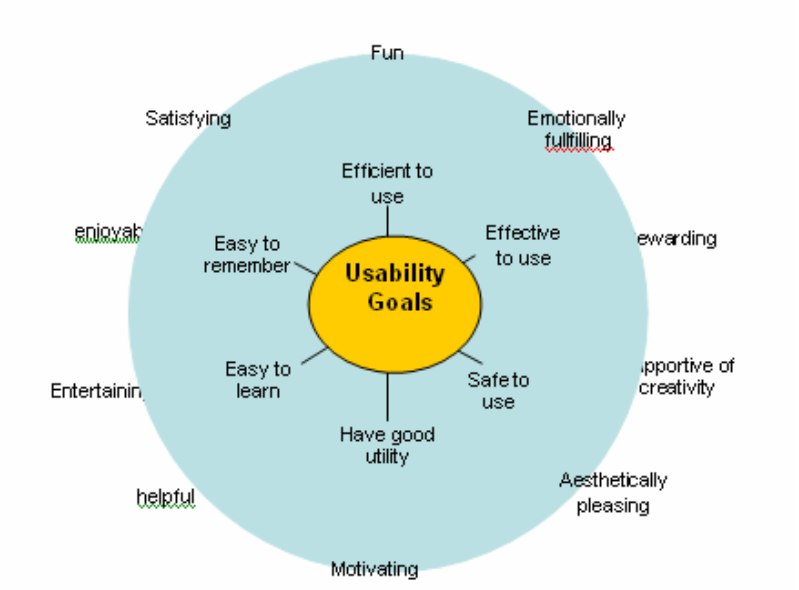
© Copyright Virtual University of Pakistan

improving efficiency and productivity at work, interaction design is increasingly concerning itself with creating systems that are:

- Satisfying
- Enjoyable
- Fun
- Entertaining
- Helpful
- Motivating
- Aesthetically pleasing
- Supportive of creativity
- Rewarding
- Emotionally fulfilling

The goals of designing interactive products to be fun, enjoyable, pleasurable, aesthetically pleasing and so on are concerned primarily with the user experience. By this we mean what the interaction with the system feels like to the users. This involves, explicating the nature of the user experience in subjective terms. For example, a new software package for children to create their own music may be designed with the primary objectives of being fun and entertaining. Hence, user experience goals differs from the more objective usability goals in that they are concerned with how user experience an interactive product from their perspective, rather than assessing how useful or productive a system is from its own perspective. The relationship between two is shown in figure.

Recognizing and understanding the trade-offs, between usability and user experience goals, is important. In particular, this enables designers to become aware of the consequences of pursuing different combinations of them in relation to fulfilling different users' needs. Obviously, not all of the usability goals and user experience goals apply to every interactive product being developed. Some combination will also be incompatible. For example, it may not be possible or desirable to design a process control system that is both safe and fun. [1]



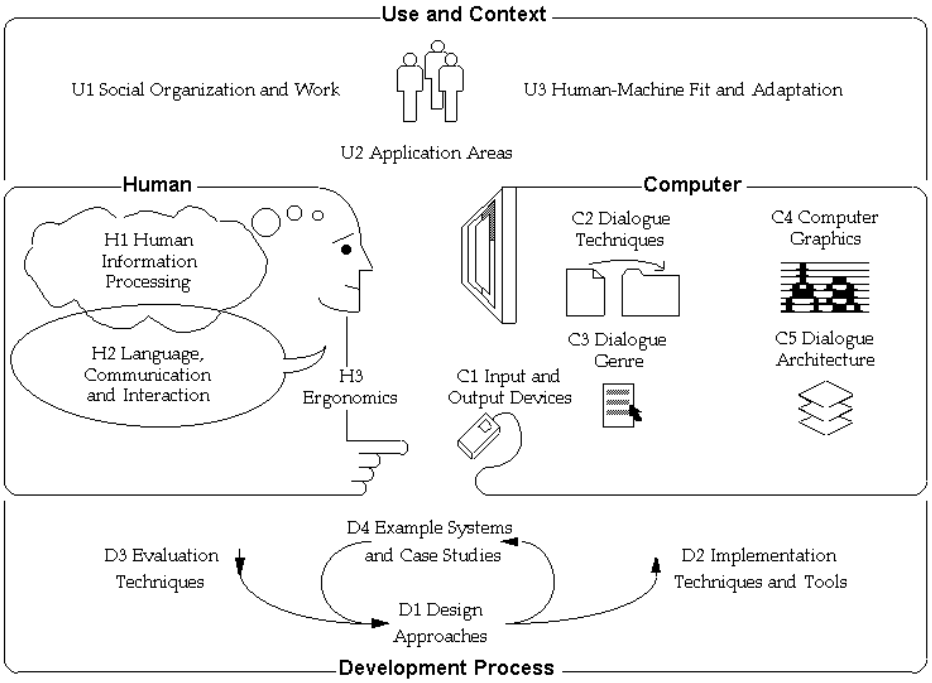
© Copyright Virtual University of Pakistan

4.2 **Evolution of HCI**

Figure shows the main topics that make up the discipline of HCI. All HCI takes place within a social and organizational context. Different kinds of applications are required for different purposes and care is needed to divide tasks between humans and machines, making sure that those activities and routine are allocated to machines.

Knowledge of human psychological and physiological abilities and, more important still their limitations is important.

As shown in figure, this involves knowing about such things as human information processing, language, communication, interaction and ergonomics. Similarly it is essential to know about the range of possibilities offered by computer hardware and software so that knowledge about humans can be mapped on to the technology appropriately. The main issues for consideration on the technology side involve input techniques, dialogue technique, dialogue genre or style, computer graphics and dialogue architecture. This knowledge has to be brought together some how into the design and development of computer systems with good HCI, as shown at the bottom of the figure. Tools and techniques are needed to realize systems. Evolution also plays an important role in this process by enabling designers to check that their ideas really are what users want.



Three systems that provide landmarks along this evolutionary path are the Dynabook, the Star and the Apple Lisa, predecessor of today's Apple Macintosh machines. An important unifying theme present in all three computer systems is that they provided a form of interaction that proved effective and easy for novices and experts alike. They were also easy to learn, and provided a visual-spatial interface whereby, in general, objects could be directly manipulated, while the system gave immediate feedback.

© Copyright Virtual University of Pakistan

Dynabook

Alan Kay designed the first object-oriented programming language in the 1970s. Called Smalltalk, the programs were the basis for what is now known as windows technology—the ability to open more than one program at a time on a personal computer. However, when he first developed the idea, personal computers were only a concept. In fact, the idea of personal computers and laptops also belongs to Kay. He envisioned the Dynabook—a notebook-sized computer, with a keyboard on the bottom and a high-resolution screen at the top.

Star

The Xerox Star was born out of PARC's creative ferment, designing an integrated system that would bring PARC's new hardware and software ideas into a commercially viable product for use in office environments. The Star drew on the ideas that had been developed, and went further in integrating them and in designing for a class of users who were far less technically knowledgeable than the engineers who had been both the creators and the prime users of many PARC systems (one of PARC's favorite mottoes was "Build what you use, use what you build.") The Star designers were challenged to make the personal computer usable for a community that did not have previous computer experience.

From today's perspective, the Star screen looks rather unremarkable, and perhaps a bit clumsy in its graphic design—a boxy model-T when compared to the highly styled look of today's Taurus or Jaguar. What is notable from a historical perspective, of course, is how much the Star does look like current screens and how little it looks like the character-based and vector-drawing screens that preceded it.



The Star (Viewpoint) screen image The Star pioneered the now-familiar constellation of icons, moveable scrollable windows, and inter-mixed text and graphic images. The widely used graphic user interfaces (GUIs) of today are all variants of this original design. (Source: Reprinted by permission from Jeff Johnson et al. Xerox Star, a retrospective. IEEE Computer 22:9 (September, 1989), p. 13.)

The visible mechanisms on the Star display were backed up with a set of design principles that grew out of a user-oriented design methodology and by a great deal of empirical testing. Several principles were central to the Star design:

© Copyright Virtual University of Pakistan

Direct manipulation

The core concept that distinguished Star (and other Alto programs) from the conventional computer interfaces of their time was the use of a bitmapped screen to present the user with direct visual representations of objects. In the Star's desktop metaphor, documents, printers, folders, collections of folders (file drawers and cabinets), in and out boxes, and other familiar office objects were depicted on the screen. To print a document, for example, the user could point (using the mouse) to the icon for the document and the icon for the printer, while using a key on the keyboard to indicate a Copy operation.

WYSIWYG (what you see is what you get)

In previously available programs for producing sophisticated graphical output—such as drawings or page layout with multiple fonts—the user created and edited a representation that looked like a programming language, and then compiled the resulting program into a visible form. Alto programs pioneered a new style that Star unified, in which the user works directly with the desired form, through direct manipulation. The user makes changes by operating on a direct representation of what will appear on the printed page. The Star user could intermix text, tables, graphs, drawings, and mathematical formulas. In fact, most of the popular microcomputer applications of today have not yet reached the degree of integration that Star offered more than a decade ago.

Consistency of commands

Because a single development group developed all Star applications in a unified way, it was possible to adhere to a coherent and consistent design language. The Star keyboard embodied a set of generic commands, which were used in a consistent way across all applications: Move, Copy, Delete, Open, Show Properties, and Same (copy properties). Evoking one of these commands produced the same behavior whether the object is being moved or copied, for example, was a word of text, a drawing element, or a folder of documents. Through the use of property sheets the user could manipulate the aspects that were specific to each element, such as the font of a text character, or the brush width of a painted line. The Open command was the basis for applying a technique of progressive disclosure—showing the user only the relevant information for a task at hand, and then providing a way to reveal more possibilities, as they were needed.

In addition to these three key concepts, many specific design features made the Star unique, including its attention to the communicative aspects of graphic design, its integration of an end-user scripting language (CUSP), and its underlying mechanisms for internationalization—from the very beginning, Star versions were developed in several languages, including non-European languages with large character sets, non–left-to-right orthography, and so on.

Some of the aspects that led to the Star's design quality may have also hampered its commercial success—in particular Xerox's dependence on development groups within a single company to produce all the applications software.

Lisa by Apple

The GUI (Graphical User Interface) that started it all. If you are sitting in front of a computer with a mouse and pull down menus you owe it to this machine. Windows proponents will tell you that Xerox PARC developed GUIs and Apple stole it from them,

© Copyright Virtual University of Pakistan

therefore what Mr. Gates has done is okay. Xerox had the core idea, but I've seen video of the early PARC work. It was advanced but it was not nearly what the Lisa (and later the Mac) became.
The first Apple Lisa was equipped with dual 5.25 inch floppy drives in addition to a huge external hard drive (shown here). The Apple Lisa 2/10 moved the hard drive inside the case. It lost one floppy drive and the Macintosh the newer 3.5-inch floppy shared the remaining one.
My Lisa is the later variety. In fact I have no way of knowing how mine was sold but the Lisa was later marketed as the Macintosh XL: a bigger sister to the little Macintosh. Lisa lacked the ROM toolbox built into every Macintosh so it had to do Macintosh emulation through a new operating system known as MacWorks. It allowed Lisa to pretend she was a Macintosh. Why do this when you could just buy a Mac? Lisa offered more RAM (1 meg) a hard drive (10 meg) and some businesses had already bought them.



While giving credit to the workers at Xerox it should also be mentioned that much of the groundwork was done in the 1960s and early 1970s. one influential researcher was Licklider (1960), who visualized a symbiotic relationship between humans and computers. He envisaged computers that would be able to do more than simply handle information: the partnership of computer and human brain would greatly enhance thinking processes and lead to more creative achievements. Another influential development was the pioneering work of Sutherland (1963), who developed the Sketchpad system at MIT. The Sketchpad system introduced a number of powerful new ideas, including the ability to display, manipulate and copy pictures represented on the screen and the use of new input devices such as the light pen.

Alongside developments in interactive graphic interface, interactive text processing systems were also evolving at a rapid rate. Following in the footsteps of line and display editors was the development of systems that allowed users to create and edit documents that were represented fully on the screen. The underlying philosophy of these systems is captured by the term WYSIWYG, which stands for ‘what you see is what you get’ (pronounced ‘whizzee-wig’). In other words, the documents were displayed on the screen exactly as they would look in printed form. This was in stark contrast to earlier document editors, where commands were embedded in the text and it was impossible to see what document would look like without printing it.

Interestingly, difference in research and development interests could be discerned on the two sides of the Atlantic. Pioneers of HCI in the USA were primarily concerned with how the computer could enrich our lives and make them easier. They foresaw it as a tool that could facilitate creativity and problem solving. In Europe, in 1980 researchers began to be more concerned with constructing theories of HCI and developing methods of design which would ensure that the needs of users and their tasks were taken into account. One

© Copyright Virtual University of Pakistan

of the major contributions from the European side was an attempt to formalize more fully the concept of usability and to show how it could be applied to the design of computer systems (Shackel, 1981).

During the technology explosion of the 1970s the notion of user interface, also known as the Man-Machine Interface (MMI), became a general concern to both system designers and researchers. Moran defined this term as 'those aspects of the system that the user comes in contact with' (1981, p.4), which in turn means 'an input language for the user, an output language for the machine, and a protocol for interaction' (Chi, 1985, p.671).

Academic researchers were concerned about how the use of computer might enrich the work and personal lives of people. In particular, they focused on the capabilities and limitations of human users, that is, understanding the 'people side' of the interaction with computer systems. At that time this primarily meant understanding people's psychological processes when interacting with computers. However, as the field began to develop it soon became clear that other aspects impinge on users and management and organizational issues and health hazards are all important factors contributing to the success or failure of using the computer systems. [2]

Reference:

[1] About Face 2.0 the essentials of interaction design by Alan Cooper
[2] Human Computer Interaction by Jenny Preece

© Copyright Virtual University of Pakistan

Lecture 5.

Discipline of Human Computer Interaction

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Describe the relationship of Usability and quality
- Understand HCI Discipline

Human-computer Interaction is the kind of discipline, which is neither the study of human, nor the study of technology, but rather the bridging between those two. So you always have to have one eye open to the question: what can the technology do? How can you build it? What are the possibilities? And one eye open to the question: what are people doing and how would this fit in? What they would do with it? If you lose sight of either of those, you fail to design well. And of course they require different ways of thinking. So I think the challenge is to keep knowledge of both the technology and the people playing off against each other in order to develop new things.

If you build something you need to consider not just 'I'm building something because I need to build it', but 'what effect is it going to have on the way people work and the way people live?'

5.1 Quality

Let us firstly look at a general definition of quality.

According to the American Heritage Dictionary "characteristic or attribute of something." As an attribute of an item, quality refers to measurable characteristics---things we are able to compare to know standards such as length, color, electrical properties, malleability, and so on.
Now as we are concerned with software quality so let us look at some other definitions:
According to British Defense Industries Quality Assurance Panel "Quality is conformance to specifications". So, according to this definition quality is the measure of degree to which the design specifications are followed during manufacturing. The greater the degree of conformance, the higher the level of quality is.
Philip Crosby describes, "Quality is conformance to requirements." Here software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.

Juran says, "Quality is fitness for purpose or use"

"Quality is a predictable degree of uniformity and dependability, at low cost and suited to the market", defined by Edward Deming.
By R J Mortiboys "Quality is synonymous with customer needs and expectations." "Quality is meeting the (stated) requirements of the customer- now and in the future." By Mike Robinson.

© Copyright Virtual University of Pakistan

"Quality is the total composite product and service characteristics of marketing, engineering, manufacturing and maintenance through which the product and service in use will meet the expectations by the customer"

(Armand Feigenbaum)

"Totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs."

All above-mentioned definitions refer quality as a conformance to requirements or conformance to specification or as a synonymous with customer needs and expectations etc. In my point of view or with respect to HCI, quality is something beyond meeting the specifications, requirements or customer expectations. For example, consider a scenario, as you know, there is always a quality assurance department in any software house which checks the final products with reference to their specification or requirements. The products that do not fulfill their specifications or requirements they are considered bugged. In my scenario, what will be the matter if the specifications or requirements, which are being used to measure quality, are not complete? That's why, I think, quality is beyond the conformance to specifications or requirements or even the customer expectations.

I think quality cannot be measured just by the requirements or specifications described by the customer rather you should approach to that end user who will use this product. The expectations or needs of the end user can be the measure of quality. So, we can say, as much as the product will be useable for end user as much higher will be its quality.
To understand the relationship of quality and usability in a software reference, look at the definition of software quality. "The extent to which a software product exhibits these characteristics"

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

© Copyright Virtual University of Pakistan

5.2 **Interdisciplinary nature of HCI**

<table>
<tr><td colspan="2">Organizational Factors
Training, job design, politics, roles Work organization</td><td>Environmental Factors
Noise, heating, ventilation,lighting</td></tr>
<tr><td>Health and Safety
Stress, headaches, Muscular-skeleton, disorders</td><td>Cognitive processes and capabilities
The User
Motivation, Enjoyment, Satisfaction, Personality
Experience level</td><td>Comfort Level
Seating Equipment layout</td></tr>
<tr><td colspan="3">User Interface
Input devices, output displays, dialogue structures, User of colour, icons, commands, graphics, natural language</td></tr>
<tr><td colspan="3">Task Factors
Easy, complex, novel, Task allocation, repetitive, Monitoring, skills, multi-media</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3">Constraints</td></tr>
<tr><td colspan="3">System Functionality
Hardware, software, application</td></tr>
<tr><td colspan="3">Productivity Factors
Increase output, increase quality, decrease costs, decrease errors, decrease labour requirements, and decrease production time, Increase creative and innovative ideas leading to new products</td></tr>
</table>

The main factors that should be taken in account in HCI design are shown in above figure. Primarily, these relate directly to users, such as comfort and health, or are concerned with users' work, the work environment or the technology being used. What makes the analysis even more complex is that many factors inevitably interact with each other. For example, if changes are made to improve productivity, this may have undesirable effects on users' motivations and levels of satisfaction because issues relating to job design and work organization are ignored.

Case Study – Ticketing System

A small travel agency with a number of shops distributed throughout the country decides that, in order to survive in the travel industry, it needs to install an efficient ticketing system. Current practice involves sales staff in a lengthy procedure for issuing tickets to customers. First they have to call an airline to check if there are any vacant seats for the time when the customer wishes to fly. Then they have to check with the customer which of the available seats is suitable before making a reservation with the airline. The ticket is then written out by hand. In addition, the customer needs a receipt and an itinerary, which are also written by hand. One of the biggest problems with this practice is getting a telephone connection to the airline. This means that customers often have to wait while a frustrated sales assistant keeps trying in vain. To overcome this problem it is common practice to ask the customers to come back later in the hope that the sales staff will manage to get through to the airline in the meantime. Another time-consuming job is accounting for each ticket that has been issued, and the sales staff has to do this by hand every two weeks.

© Copyright Virtual University of Pakistan

Before deciding to get new system the branch manager does some background research into how the agency really functions. She starts by visiting branches in a sister company that is using a computerized ticketing system. After talking to the staff for just a short time she discovers that there are problems. The sales staff complains that the computer is always going wrong and that they don't trust it. Furthermore, they can't understand some of the messages that it produces when they make errors. In fact, they wish they could go back to the old un-computerized way of working. Sales figures since the new system was installed are also disappointing and a large number of staff have left the office. Not surprisingly, the manager is consultants examine the users' needs and how they currently go about their work in detail and also find out exactly what the goals of the company are. They then recommend a system with the following characteristics:

- Immediate ticket booking via a computer connection (alleviating the problem of engaged phone line),
- Automatic print-out of tickets, itineraries and receipts (eliminating the need to write these by hand and thereby reducing the possibility of errors and illegibility while speeding up the process),
- Direct connection between the booking system and accounting (speeding up the process of accounting),
- Elimination of booking forms (reducing overheads as less paper and time are used).

The consultants suggest making the interface to the system mimic the non-computerized task, so menus and forms are used, which means that the sales assistant only has to select options and fill in the resulting forms by typing at a keyboard.

The consultants are optimistic that customer satisfaction will improve because customer will get their tickets on the spot. They point out to the manager, however, that in order to get the most out of the new system the layout of the agency needs to be changed to make it comfortable for the sales staff to operate the compute, while still providing scope for direct contact with customers. Staff will also need training, and some careful changes to existing jobs are needed too—job design. In particular, technology means that they will need support during the period of change. Staff will also need to know how to cope when an airline's computer malfunctions. Changes in employment conditions must also be examined. For instance, if staff is expected to carry out more transactions in less time, are they going to be rewarded for this extra activity? Staff relations with other staff in the company who will not be using the computerized system must also be taken into account. For example, problems associated with technology power such as feelings f elitism among staff that know how to use the new technology, will need to be resolved.

HCI understands the Complex Relationship between Human and Computers, which are two distinct 'Species'. Successful Integration is dependent upon a better understanding of both Species. Hence HCI borrows and establishes its roots in Disciplines concerned with both.

Human

- Cognitive Psychology
- Social Organizational Psychology

© Copyright Virtual University of Pakistan

- Ergonomics and Human Factors
- Linguistics
- Philosophy
- Sociology
- Anthropology

Machine

- Computer Science
- Artificial Intelligence

Other

- Engineering
- Design

Cognitive Psychology

Psychology is concerned primarily with understanding human behavior and the mental processes that underlie it. To account for human behavior, cognitive psychology has adopted the notion of information processing. Everything we see, feel, touch, taste, smell and do is couched in terms of information processing. The objective cognitive psychology has been to characterize these processes in terms of their capabilities and limitations. [2]

Social and Organizational psychology

Social psychology is concerned with studying the nature and causes of human behavior in a social context. Vaske and Grantham identify the four core concerns of social psychology as:

- The influence of one individual on another person's attitudes and behavior
- The impact of a group on its members' attitude and behavior
- The impact of a member on a group's activities and structure
- The relationship between the structure and activities of different groups.

The role of social and organizational psychology is to inform designers about social and organizational structures and about how the introduction of computers will influence working practices. [2]

Ergonomics or human factor

Ergonomics, or human factor, developed from the interests of a number of different disciplines primarily during World War II. Its purpose is to define and design tools and various artifacts for different work, leisure and domestic environments to suit the capabilities and capacities of users.

The role of ergonomist is to translate information from the above sciences into the context of design, whether for a car seat or a computer system. The objective is to maximize an operator's safety, efficiency and reliability of performance, to make a task easier, and to increase feelings of comfort and satisfaction. [2]

© Copyright Virtual University of Pakistan

Linguistics

Linguistics is the scientific study of language (Lyons, 1970). From the point of view of HCI there are several issues that may be better understood by applying knowledge and theories from linguistics. For example, in the early days of command languages there was some debate about whether or not the object to which a command applied should come before or after the command itself. When deleting a file called 'xyz', for example, should you type delete 'xyz' or 'xyz' delete. [2]

Philosophy, Sociology and Anthropology

A major concern of these disciplines until relatively recently has been to consider the implication of the introduction of IT to society. More recently, attempts are being made to apply methods developed in the social sciences to the design and evaluation of systems. The reason for applying social science methods of analysis to HCI, it is argued, are that a more accurate description of the interaction between users, their work, the technology that they use and the environment in which they are situated can be obtained. One application of social science methods has been to characterize computer supported cooperative writing (CSCW), which is concerned with sharing software and hardware among groups of people working together. The is to design tools and ways of working which optimize the shared technology so that maximum benefit can be obtained by all those who use or are affected by it. [2]

Artificial Intelligence

Artificial Intelligence (AI) is concerned with the design of intelligent computer programs which simulate different aspects of intelligent human behavior. The relationship of AI to HCI is mainly concerned with user's needs when interacting with an intelligent interface. These include, for example, the use of natural language and speech as a way of communicating with a system and the need for system to explain and justify its advice. [2]

Computer Science

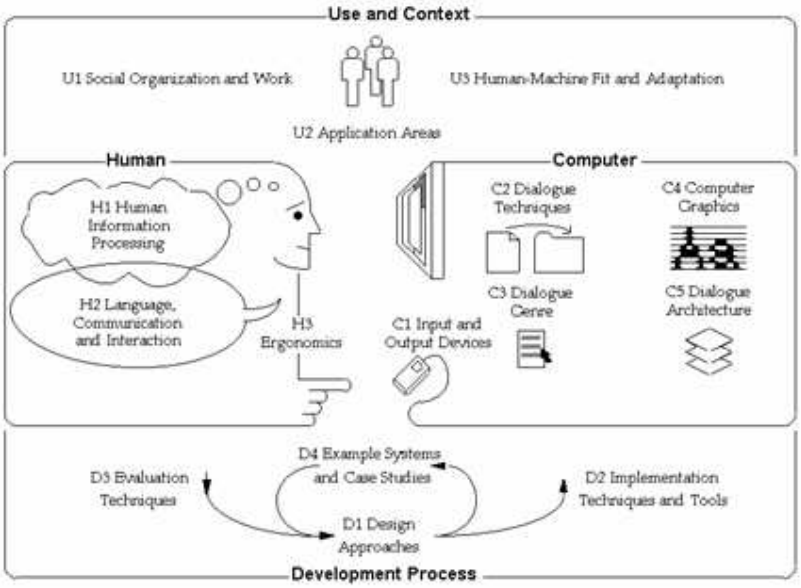
One of the main contributions of computer science to HCI is to provide knowledge about the capabilities of technology and ideas about how this potential can be harnessed. In addition, computer scientists have been concerned about developing various kinds of techniques to support software design, development and maintenance. In particular, there has been a strong interest in automating design and development when feasible. [2]

Engineering and design

Engineering is an applied science, which relies heavily on model building and empirical testing. Design contributes creative skills and knowledge to this process. In many respects the greatest influence of engineering on HCI and subsequently on interface and system development is through software engineering.

Design too is a well-established discipline in its own right, which has potential benefits when applied to HCI problems. An obvious example is graphic design.[2]

© Copyright Virtual University of Pakistan



Reference:

Human-Computer Interaction by Jenny Preece
Software Engineering A Practitioner's Approach by Roger S. Pressman
Definitions of Quality - Sandeep's Quality Page.htm

© Copyright Virtual University of Pakistan

Lecture 6.

Cognitive Frameworks

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the importance of Cognition
- Understand different cognitive frameworks in HCI

6.1 Introduction

Imagine trying to drive a car by using just a computer keyboard. The four arrow keys are used for steering, the space bar for braking, and the return key for accelerating. To indicate left you need to press the F1 key and to indicate right the F2 key. To sound your horn you need to press the F3 key. To switch the headlights on you need to use the F4 key and, to switch the windscreen wipers on, the F5 key. Now imagine as you are driving along a road a ball is suddenly kicked in front of you. What would you do? Bash the arrow keys and the space bar madly while pressing the F4 key? How would rate your chance of missing the ball?

Most of us would bald at the very idea of driving a car this way. Many early video games, however, were designed along these lines: the user had to press an arbitrary combination of function keys to drive or navigate through the game. More recently, computer consoles have been designed with the user's capabilities and demands of the activity in ming. Much better way of controlling and interacting, such as through using joysticks and steering wheels are provided that map much better onto the physical and cognitive aspects of driving and navigating.

We have to understand the limitations of the people to ease them. Let us see what cognitive psychology is and how it helps us.

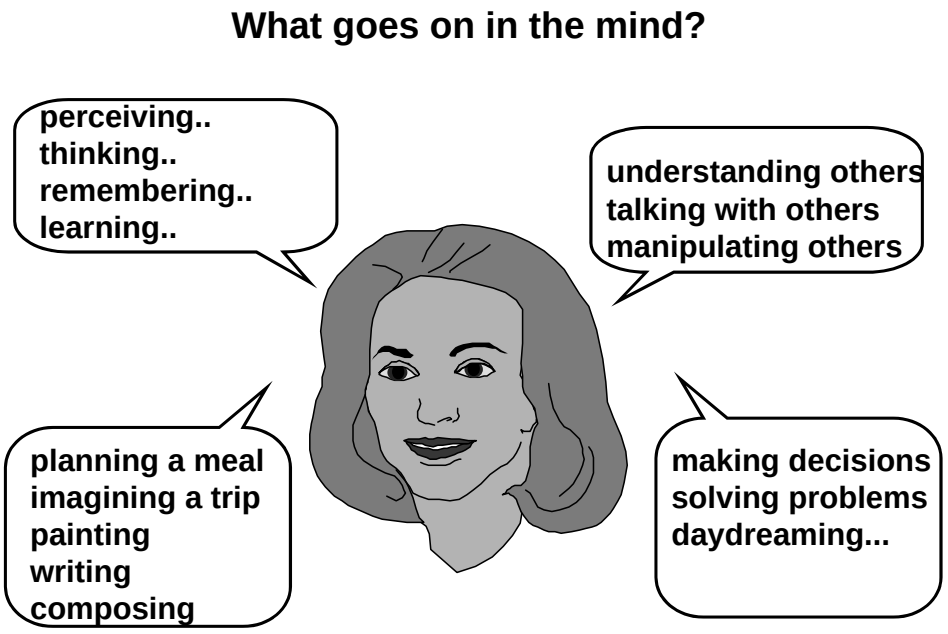
Cognitive Psychology

Psychology is concerned primarily with understanding human behavior and the mental processes that underlie it. To account for human behavior, cognitive psychology has adopted the notion of information processing. Everything we see, feel, touch, taste, smell and do is couched in terms of information processing. The objective cognitive psychology has been to characterize these processes in terms of their capabilities and limitations. For example, one of the major preoccupations of cognitive psychologists in the 1960s and 1970s was identifying the amount o f information that could be processed and remembered at any one time. Recently, alternative psychological frameworks have been sought which more adequately characterize the way people work with each other and with the various artifacts, including computers, that they have use. Cognitive psychology have attempted to apply relevant psychological principles to HCI by using a variety of methods, including development of guidelines, the use of models to predict human performance and the use of empirical methods for testing computer systems.

© Copyright Virtual University of Pakistan

Cognition

The dominant framework that has characterized HCI has been cognitive. Let us define cognition first:
Cognition is what goes on in out heads when we carry out our everyday activities.
In general, cognition refers to the processes by which we become acquainted with things or, in other words, how we gain knowledge. These include understanding, remembering, reasoning, attending, being aware, acquiring skills and creating new ideas.
As figure indicates there are different kinds of cognition.



The main objective in HCI has been to understand and represent how human interact with computers in term of how knowledge is transmitted between the two. The theoretical grounding for this approach stems from cognitive psychology: it is to explain how human beings achieve the goals they set.
Cognition has also been described in terms of specific kinds of processes. These include:

- Attention
- Perception and recognition
- Memory
- Learning
- Reading, speaking, and listening
- Problem solving, planning, reasoning, decision-making.

It is important to note that many of these cognitive processes are interdependent: several may be involved for a given activity. For example, when you try to learn material for an exam, you need to attend the material, perceive, and recognize it, read it, think about it,

© Copyright Virtual University of Pakistan

and try to remember it. Thus cognition typically involves a range of processes. It is rare for one to occur in isolation.

6.2 Modes of Cognition

Norman (1993) distinguishes between two general modes:

1. Experiential cognition
2. Reflective cognition

Experiential cognition

It is the state of mind in which we perceive, act, and react to events around us effectively and effortlessly. It requires reaching a certain level of expertise and engagement. Examples include driving a car, reading a book, having a conversation, and playing a video game.

Reflective cognition

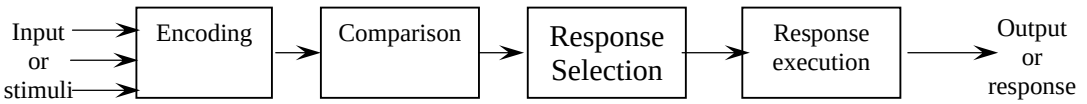
Reflective cognition involves thinking, comparing, and decision-making. This kind of cognition is what leads to new ideas and creativity. Examples include designing, learning, and writing a book.

Norman points out that both modes are essential for everyday life but that each requires different kinds of technological support.

Information processing

One of the many other approaches to conceptualizing how the mind works, has been to use metaphors and analogies. A number of comparisons have been made, including conceptualizing the mind as a reservoir, a telephone network, and a digital computer. One prevalent metaphor from cognitive psychology is the idea that the mind is an information processor.

During the 1960s and 1970s the main paradigm in cognitive psychology was to characterize humans as information processors; everything that is sensed (sight, hearing, touch, smell, and taste) was considered to be information, which the mind processes. Information is thought to enter and exit the mind through a series of ordered processing stages. As shown in figure, within these stages, various processes are assumed to act upon mental representations. Processes include comparing and matching. Mental representations are assumed to comprise images, mental models, rules, and other forms of knowledge.



Stage1 encodes information from the environment into some form of internal representation. In stage 2, the internal representation of the stimulus is compared with memorized representations that are stored in the brain. Stage 3 is concerned with deciding on a response to the encoded stimulus. When an appropriate match is made the process passes on to stage 4, which deals with the organization of the response and the necessary action. The model assumes that information is unidirectional and sequential and that each

© Copyright Virtual University of Pakistan

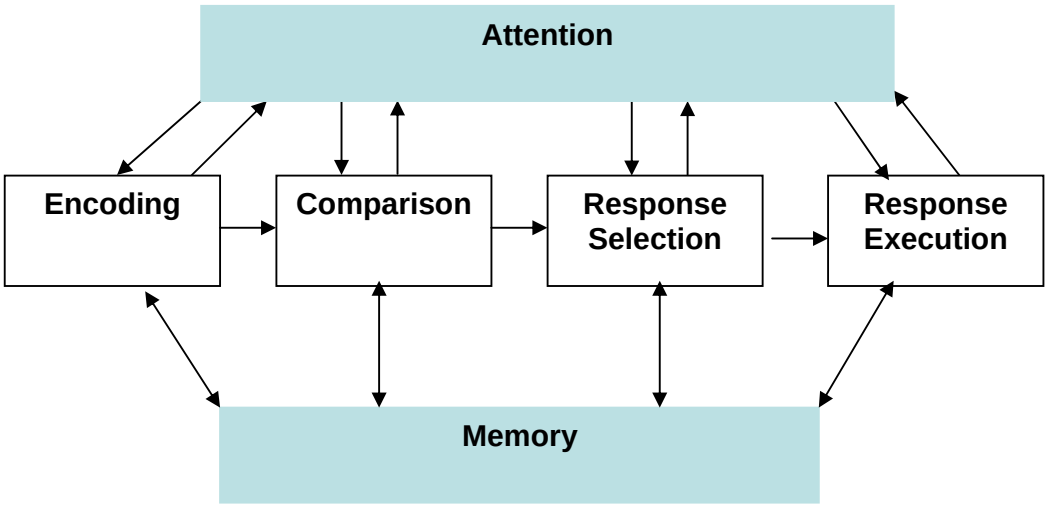
of the stages takes a certain amount of time, generally thought to depend on the complexity of the operation performed.

To illustrate the relationship between the different stages of information processing, consider the sequence involved in sending mail. First, letters are posted in a mailbox. Next, a postman empties the letters from the mailbox and takes them to central sorting office. Letters are then sorted according to area and sent via rail, road, air or ship to their destination. On reaching their destination, the letters are further forted into particular areas and then into street locations and so on. A major aspect of an information processing analysis, likewise, is tracing the mental operations and their outcomes for a particular cognitive task. For example, let us carry out an information processing analysis for the cognitive task of determining the phone number of a friend.

Firstly, you must identify the words used in the exercise. Then you must retrieve their meaning. Next you must understand the meaning of the set of words given in the exercise. The next stage involves searching your memory for the solution to the problem. When you have retrieved the number in memory, you need to generate a plan and formulate the answer into a representation that can be translated into a verbal form. Then you would need to recite the digits or write them down.

Extending the human information processing model

Two main extensions of the basic information-processing model are the inclusion of the processes of attention and memory. Figure shows the relationship between the different processes. [3]



In the extended model, cognition is viewed in terms of:

1. How information is perceptual processors
2. How that information is attended to, and
3. How that information is processed and stored in memory.

6.3 Human processor model

The information-processing model provides a basis from which to make predictions about human performance. Hypotheses can be made about how long someone will take to

© Copyright Virtual University of Pakistan

perceive and responds to a stimulus (also known as reaction time) and what bottlenecks occur if a person is overloaded with too much information. The best-known approach is the human processor model, which models the cognitive processes of a user interacting with a computer. Based on the information-processing model, cognition is conceptualized as a series of processing stages where perceptual, cognitive, motor processors are organized in relation to one another. The model predicts which cognitive processes are involved when a user interacts with a computer, enabling calculations to be made how long a user will take to carry out various tasks. This can be very useful when comparing different interfaces. For example, it has been used to compare how well different word processors support a range of editing tasks.

The information processing approach is based on modeling mental activities that happen exclusively inside the head. However, most cognitive activities involve people interacting with external kinds of representations, like books, documents, and computers—not to mentions one another. For example, when we go home from wherever we have been we do not need to remember the details of the route because we rely on cues in the environment (e.g., we know to turn left at the red house, right when the road comes to a T-junction, and so on.). Similarly, when we are at home we do not have to remember where everything is because information is "out there." We decide what to eat and drink by scanning he items in the fridge, find out whether any messages have been left by glancing at the answering machine to see if there is a flashing light, and so on. [2]

6.4 GOMS

Card et al. have abstracted a further family of models, known as GOMS (goals, operations, methods and selection rules) that translate the qualitative descriptions into quantitative measures. The reason for developing a family of models is that it enables various qualitative and quantitative predictions to be made about user performance.

Goals

These are the user's goals, describing what the user wants to achieve. Further, in GOMS the goals are taken to represent a 'memory point' for the user, from which he can evaluate what should be done and to which he may return should any errors occur. [1]

Operators

These are the lowest level of analysis. They are the basic actions that the user must perform in order to use the system. They may affect the system (e.g., press the 'X' key) or only the user's mental state (e.g., read the dialogue box). There is still a degree of flexibility about the granularity of operators; we may take the command level "issue the select command" or be more primitive; "move mouse to menu bar, press center mouse button…." [1]

Methods

As we have already noted, there are typically several ways in which a goal can be split into sub goals. [1]

Selection

Selection means of choosing between competing methods [1]

One of the problems of abstracting a quantitative model from a qualitative description of user performance is ensuring that two are connected. In particular, it a has been noted that

© Copyright Virtual University of Pakistan

the form and contents of GOMS family of models are relatively unrelated to the form and content of the model human processor and it also oversimplified human behavior. More recently, attention has focused on explaining:

- Knowledge Representation Models
  How knowledge is represented

- Mental Models
  How mental models (these refer to representation people construct in their mind of themselves, others, objects and the environment to help them know what to do in current and future situations) develop and are used in HCI

- User Interaction Learning Models
  How user learn to interact and become experienced in using computer system.

With respect to applying this knowledge to HCI design, there has been considerable research in developing:

Conceptual Models

Conceptual models are (these are the various ways in which systems are understood by different people) to help designers develop appropriate interfaces.

Interface Metaphor

Interface metaphors are (these are GUIs that consists of electronic counterparts to physical objects in the real world) to match the knowledge requirements of users.

6.5 Recent development in cognitive psychology

With the development of computing, the activity of brain has been characterized as a series of programmed steps using the computer as a metaphor. Concept such as buffers, memory stores and storage systems, together with the type of process that act upon them (such as parallel verses serial, top-down verses down-up) provided psychologist with a mean of developing more advanced models of information processing, which was appealing because such models could be tested. However, since the 1980s there has been a more away from the information-processing framework with in cognitive psychology. This has occurred in parallel with the reduced importance of the model human processor with in HCI and the development other theoretical approaches. Primarily, these are the computational and the connectionist approaches. More recently other alternative approaches have been developed that has situated cognitive activity in the context in which they occur. [3]

Computational Approaches

Computational approaches continue to adopt the computer metaphor as a theoretical framework, but they no longer adhere to the information-processing framework. Instead, the emphasis is on modeling human performance in terms of what is involved when information is processed rather than when and how much. Primarily, computational models conceptualize the cognitive system in terms of the goals, planning and action that are involved in task performance. These aspects include modeling: how information is organized and classified, how relevant stored information is retrieved, what decisions are made and how this information is reassemble. Thus tasks are analyzed not in terms of the

© Copyright Virtual University of Pakistan

amount of information processed per se in the various stages but in terms of how the system deals with new information. [3]

Connectionist Approaches

Connectionist approaches, otherwise known as neural networks or parallel distributed processing, simulate behavior through using programming models. However, they differ from conceptual approaches in that they reject the computer metaphor as a theoretical framework. Instead, they adopt the brain metaphor, in which cognition is represented at the level of neural networks consisting of interconnected nodes. Hence all cognitive processes are viewed as activations of the nodes in the network and the connections between them rather than the processing and manipulation of information. [3]

6.6 External Cognition

External cognition is concerned with explaining the cognitive processes involved when we interact with different external representations. A main goal is to explicate the cognitive benefits of using different representations for different cognitive activities and the processes involved. The main one includes:

1. externalizing to reduce memory load
2. computational offloading
3. Annotating and cognitive tracing.

Externalizing to reduce memory load

A number of strategies have been developed for transforming knowledge into external representations to reduce memory load. One such strategy is externalizing things we find difficult to remember, such as birthdays, appointments and addresses.
Externalizing, therefore, can help reduce people's memory burden by:

- reminding them to do something (e.g., to get something for their mother's birthday)
- reminding them of what to do (e.g., to buy a card)
- reminding them of when to do something (send it by a certain date)

Computational offloading

Computational offloading occurs when we use a tool or device in conjunction with an external representation to help us carry out a computation. An example is using pen or paper to solve a math problem.[2]

Annotating and cognitive tracing

Another way in which we externalize our cognitions is by modifying representations to reflect changes that are taking place that we wish to mark. For example, people often cross thinks off in to-do list to show that they have been completed. They may also reorder objects in the environment; say by creating different piles as the nature of the work to be done changes. These two kinds of modification are called annotating and cognitive tracing:

© Copyright Virtual University of Pakistan

- Annotating involves modifying external representations, such as crossing off underlining items.
- Cognitive tracing involves externally manipulating items different orders or structures.

Information Visualization

A general cognitive principle for interaction design based on the external cognition approach is to provide external representations at the interface that reduce memory load and facilities computational offloading. Different kinds of information visualizations can be developed that reduce the amount of effort required to make inferences about a given topic (e.g., financial forecasting, identifying programming bugs). In so doing, they can extend or amplify cognition, allowing people to perceive and do activities tat they couldn't do otherwise. [2]

6.7 Distributed cognition

Distributed cognition is an emerging theoretical framework whose goal is to provide an explanation that goes beyond the individual, to conceptualizing cognitive activities as embodied and situated within the work context in which they occur. Primarily, this involves describing cognition as it is distributed across individuals and the setting in which it takes place. The collection of actors (more generally referred to just as 'people' in other parts of the text), computer systems and other technology and their relations to each other in environmental setting in which they are situated are referred to as functional systems. The functional systems that have been studied include ship navigation, air traffic control, computer programmer teams and civil engineering practices.

A main goal of the distributed cognition approach is to analyze how the different components of the functional system are coordinated. This involves analyzing how information is propagated through the functional system in terms of technological cognitive, social and organizational aspects. To achieve this, the analysis focuses on the way information moves and transforms between different representational states of the objects in the functional system and the consequences of these for subsequent actions.[3]

References:

[1] Human Computer Interaction by Alan Dix
[2] Interaction Design by Jenny Preece
[3] Human Computer Interaction by Jenny Preece

© Copyright Virtual University of Pakistan

Lecture 7.

Human Input-Output Channels – Part I

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand role of input-output channels
- Describe human eye physiology and
- Discuss the visual perception

7.1 Input Output channels

A person's interaction with the outside world occurs through information being received and sent: input and output. In an interaction with a computer the user receives information that is output by the computer, and responds by providing input to the computer – the user's output become the computer's input and vice versa. Consequently the use of the terms input and output may lead to confusion so we shall blur the distinction somewhat and concentrate on the channels involved. This blurring is appropriate since, although a particular channel may have a primary role as input or output in the interaction, it is more than likely that it is also used in the other role. For example, sight may be used primarily in receiving information from the computer, but it can also be used to provide information to the computer, for example by fixating on a particular screen point when using an eye gaze system.

Input in human is mainly though the senses and out put through the motor control of the effectors. There are five major senses:

- Sight
- Hearing
- Touch
- Taste
- Smell

Of these first three are the most important to HCI. Taste and smell do not currently play a significant role in HCI, and it is not clear whether they could be exploited at all in general computer systems, although they could have a role to play in more specialized systems or in augmented reality systems. However, vision hearing and touch are central.

Similarly there are a number of effectors:

- Limbs
- Fingers
- Eyes
- Head

© Copyright Virtual University of Pakistan

- Vocal system.

In the interaction with computer, the fingers play the primary role, through typing or mouse control, with some use of voice, and eye, head and body position.

Imagine using a personal computer with a mouse and a keyboard. The application you are using has a graphical interface, with menus, icons and windows. In your interaction with this system you receive information primarily by sight, from what appears on the screen. However, you may also receive information by ear: for example, the computer may 'beep' at you if you make a mistake or to draw attention to something, or there may be a voice commentary in a multimedia presentation. Touch plays a part too in that you will feel the keys moving (also hearing the 'click') or the orientation of the mouse, which provides vital feedback about what you have done. You yourself send information to the computer using your hands either by hitting keys or moving the mouse. Sight and hearing do not play a direct role in sending information in this example, although they may be used to receive information from a third source (e.g., a book or the words of another person) which is then transmitted to the computer.

7.2 Vision

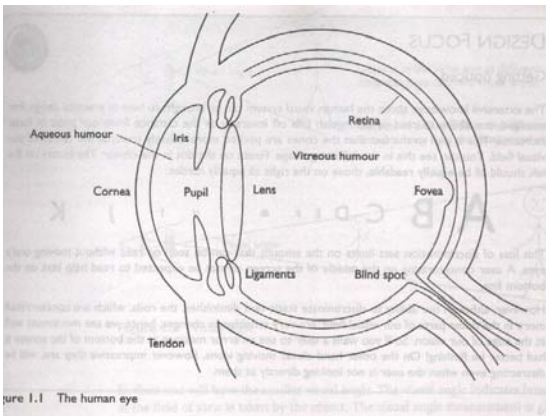
Human vision is a highly complex activity with range of physical and perceptual limitations, yet it is the primary source of information for the average person. We can roughly divide visual perception into two stages:

- the physical reception of the stimulus from outside world, and
- The processing and interpretation of that stimulus.

On the one hand the physical properties of the eye and the visual system mean that there are certain things that cannot be seen by the human; on the other interpretative capabilities of visual processing allow images to be constructed from incomplete information. We need to understand both stages as both influence what can and can not be perceived visually by a human being, which is turn directly affect the way that we design computer system. We will begin by looking at the eye as a physical receptor, and then go onto consider the processing involved in basic vision.

The human eye

Vision begins with light. The eye is a mechanism for receiving light and transforming it into electrical energy. Light is reflected from objects in the world and their image is focused upside down on the back of the eye. The receptors in the eye transform it into electrical signals, which are passed to brain.



ure 1.1 The human eye

© Copyright Virtual University of Pakistan

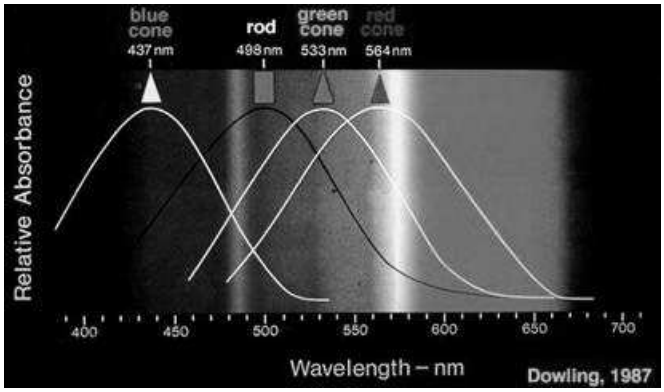
The eye has a number of important components as you can see in the figure. Let us take a deeper look. The cornea and lens at the front of eye focus the light into a sharp image on the back of the eye, the retina. The retina is light sensitive and contains two types of photoreceptor: rods and cones.

Rods

Rods are highly sensitive to light and therefore allow us to see under a low level of illumination. However, they are unable to resolve fine detail and are subject to light saturation. This is the reason for the temporary blindness we get when moving from a darkened room into sunlight: the rods have been active and are saturated by the sudden light. The cones do not operate either as they are suppressed by the rods. We are therefore temporarily unable to see at all. There are approximately 120 million rods per eye, which are mainly situated towards the edges of the retina. Rods therefore dominate peripheral vision.

Cones

Cones are the second type of receptor in the eye. They are less sensitive to light than the rods and can therefore tolerate more light. There are three types of cone, each sensitive to a different wavelength of light. This allows color vision. The eye has approximately 6 million cones, mainly concentrated on the fovea.



Fovea

Fovea is a small area of the retina on which images are fixated.

Blind spot

Blind spot is also situated at retina. Although the retina is mainly covered with photoreceptors there is one blind spot where the optic nerve enter the eye. The blind spot has no rods or cones, yet our visual system compensates for this so that in normal circumstances we are unaware of it.

Nerve cells

The retina also has specialized nerve cells called ganglion cells. There are two types:

© Copyright Virtual University of Pakistan

X-cells

These are concentrated in the fovea and are responsible for the early detection of pattern.

Y-cells

These are more widely distributed in the retina and are responsible for the early detection of movement. The distribution of these cells means that, while we may not be able to detect changes in pattern in peripheral vision, we can perceive movement.

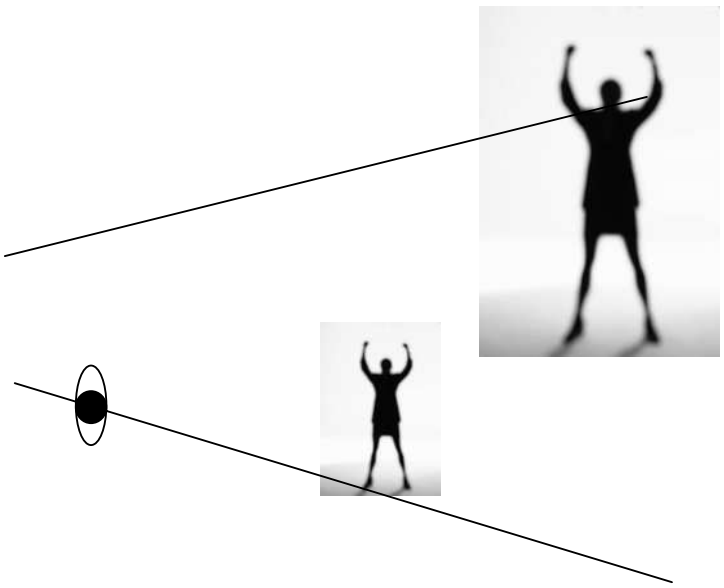
7.3 Visual perception

Understanding the basic construction of the eye goes some way to explaining the physical mechanism of vision but visual perception is more than this. The information received by the visual apparatus must be filtered and passed to processing elements which allow us to recognize coherent scenes, disambiguate relative distances and differentiate color. Let us see how we perceive size and depth, brightness and color, each of which is crucial to the design of effective visual interfaces.

Perceiving size and depth

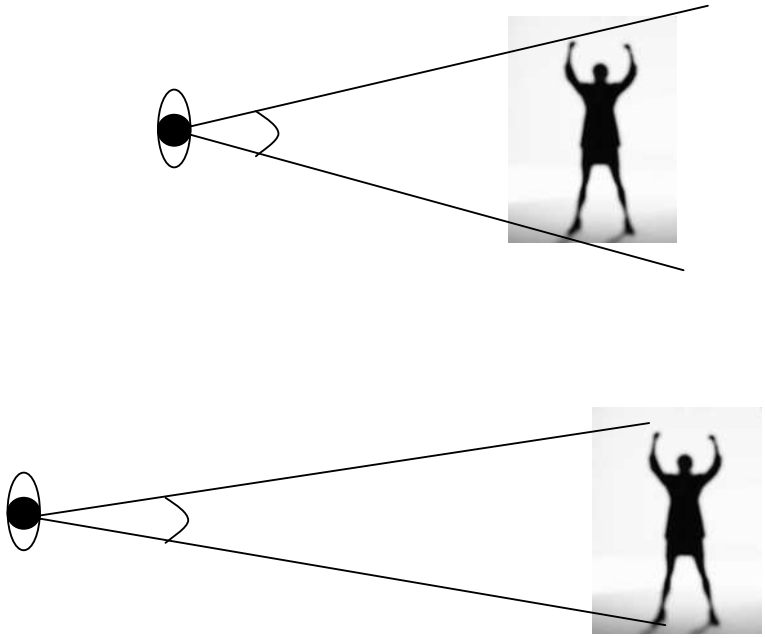
Imagine you are standing on a hilltop. Beside you on the summit you can see rocks, sheep and a small tree. On the hillside is a farmhouse with outbuilding and farm vehicles. Someone is on the track, walking toward the summit. Below in the valley is a small market town.

Even in describing such a scene the notions of size and distance predominate. Our visual system is easily able to interpret the images, which it receives to take account of these things. We can identify similar objects regardless of the fact that they appear to us to be vastly different sizes. In fact, we can use this information to judge distance.
So how does the eye perceive size, depth and relative distances? To understand this we must consider how the image appears on the retina. As we mentioned, reflected light from the object forms an upside-down image on the retina. The size of that image is specified as visual angle. Figure illustrates how the visual angle is calculated.



© Copyright Virtual University of Pakistan

If were to draw a line from the top of the object to a central point on the front of the eye and a second line from the bottom of the object to the same point, the visual angle of the object is the angle between these two lines. Visual angle is affected by both the size of the object and its distance from the eye. Therefore if two objects are at the same distance, the larger one will have the larger visual angle. Similarly, if two objects of the same size are placed at different distances from the eye, the furthest one will have the smaller visual angle, as shown in figure.



Visual angle indicates how much of the field of view is taken by the object. The visual angle measurement is given in either degrees or minutes of arc, where 1 degree is equivalent to 60 minutes of arc, and 1 minute of arc to 60 seconds of arc.

Visual acuity

So how does an object's visual angle affect our perception of its size? First, if the visual angle of an object is too small we will be unable to perceive it at all. Visual acuity is the ability of a person to perceive fine detail. A number of measurements have been established to test visual acuity, most of which are included in standard eye tests. For example, a person with normal vision can detect a single line if it has a visual angle of 0.5 seconds of arc. Spaces between lines can be detected at 30 seconds to 1 minute of visual arc. These represent the limits of human visual perception.

Law of size constancy

Assuming that we can perceive the object, does its visual angle affect our perception of its size? Given that the visual angle of an object is reduced, as it gets further away, we might expect that we would perceive the object as smaller. In fact, our perception of an object's size remains constant even if its visual angel changes. So a person's height I perceived as constant even if they move further from you. This is the law of size constancy, and it indicated that our perception of size relies on factors other than the visual angle.

© Copyright Virtual University of Pakistan

One of these factors is our perception of depth. If we return to the hilltop scene there are a number of cues, which can use to determine the relative positions and distances of the objects, which we see. If objects overlap, the object that is partially covered is perceived to be in the background, and therefore further away. Similarly, the size and height of the object in our field of view provides a cue to its distance. A third cue is familiarity: if we expect an object to be of a certain size then we can judge its distance accordingly.

Perceiving brightness

A second step of visual perception is the perception of brightness. Brightness is in fact a subjective reaction to level of light. It is affected by luminance, which is the amount of light emitted by an object. The luminance of an object is dependent on the amount of light falling on the object's surface and its reflective prosperities. Contrast is related to luminance: it is a function of the luminance of an object and the luminance of its background.

Although brightness is a subjective response, it can be described in terms of the amount of luminance that gives a just noticeable difference in brightness. However, the visual system itself also compensates for changes in brightness. In dim lighting, the rods pre-dominate vision. Since there are fewer rods on the fovea, object in low lighting can be seen easily when fixated upon, and are more visible in peripheral vision. In normal lighting, the cones take over.

Visual acuity increases with increased luminance. This may be an argument for using high display luminance. However, as luminance increases, flicker also increases. The eye will perceive a light switched on and off rapidly as constantly on. But if the speed of switching is less than 50 Hz then the light is perceived to flicker. In high luminance flicker can be perceived at over 50 Hz. Flicker is also more noticeable in peripheral vision. This means that the larger the display, the more it will appear to flicker.

Perceiving color

A third factor that we need to consider is perception of color. Color is usually regarded as being made up of three components:

- hue
- intensity
- saturation

Hue

Hue is determined by the spectral wavelength of the light. Blues have short wavelength, greens medium and reds long. Approximately 150 different hues can be discriminated by the average person.

Intensity

Intensity is the brightness of the color.

Saturation

Saturation is the amount of whiteness in the colors.

© Copyright Virtual University of Pakistan

By varying these two, we can perceive in the region of 7 million different colors. However, the number of colors that can be identified by an individual without training is far fewer.

The eye perceives color because the cones are sensitive to light of different wavelengths. There are three different types of cone, each sensitive to a different color (blue, green and red). Color vision is best in the fovea, and worst at the periphery where rods predominate. It should also be noted that only 3-4 % of the fovea is occupied by cones which are sensitive to blue light, making blue acuity lower.

Finally, we should remember that around 8% of males and 1% of females suffer from color blindness, most commonly being unable to discriminate between red and green.

The capabilities and limitations of visual processing

In considering the way in which we perceive images we have already encountered some of the capabilities and limitations of the human visual processing system. However, we have concentrated largely on low-level perception. Visual processing involves the transformation and interpretation of a complete image, from the light that is thrown onto the retina. As we have already noted, our expectations affect the way an image is perceived. For example, if we know that an object is a particular size, we will perceive it as that size no matter how far it is from us.

Visual processing compensates for the movement of the image on the retina which occurs as we around and as the object which we see moves. Although the retinal image is moving, the image that we perceive is stable. Similarly, color and brightness of objects are perceived as constant, in spite of changes in luminance.

This ability to interpret and exploit our expectations can be used to resolve ambiguity. For example consider the image shown in figure ‘a’. What do you perceive? Now consider figure ‘b’ and ‘c’. the context in which the object appears allow our expectations to clearly disambiguate the interpretation of the object, as either a B or 13.

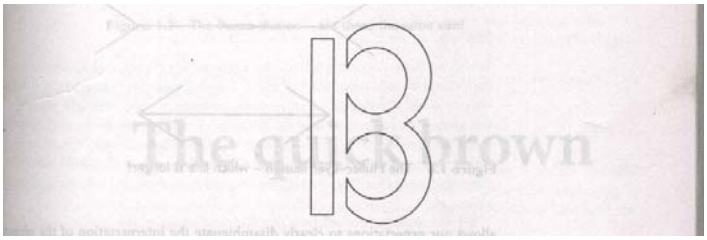
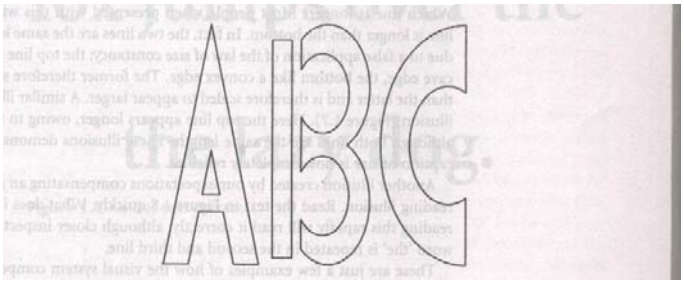
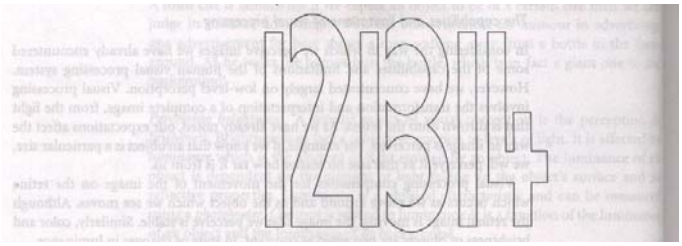


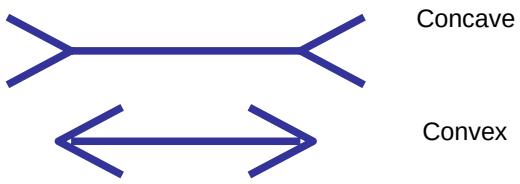
Figure a



© Copyright Virtual University of Pakistan



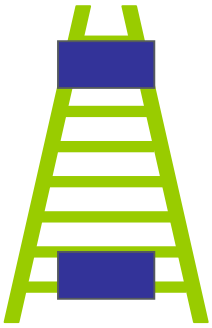
However, it can also create optical illusions. Consider figure'd'. Which line is longer?



the Muller Lyer illusion

Figure d

A similar illusion is the Ponzo illusion as shown in figure



Another illusion created by our expectations compensating an image is the proofreading illusion. Example is shown below

The quick brown fox jumps over the lazy dog.

The way that objects are composed together will affect the way we perceive them, and we do not perceive geometric shapes exactly as they are drawn. For example, we tend to magnify horizontal lines and reduce vertical. So a square needs to be slightly increased in height to appear square and line will appear thicker if horizontal rather than vertical. Optical illusions also affect page symmetry. We tend to see the center of a page as being a little above the actual center – so if a page is arranged symmetrically around the actual center, we will see it as too low down. In graphic design this is known as the optical center.

© Copyright Virtual University of Pakistan

Lecture 8.

Human Input-Output Channels Part II

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand role of color theory in design
- Discuss hearing perception
- Discuss haptic perception
- Understand movement

8.1 Color Theory

Color theory encompasses a multitude of definitions, concepts and design applications. All the information would fill several encyclopedias. As an introduction, here are a few basic concepts.

The Color Wheel



A color circle, based on red, yellow and blue, is traditional in the field of art. Sir Isaac Newton developed the first circular diagram of colors in 1666. Since then scientists and artists have studied and designed numerous variations of this concept. Differences of opinion about the validity of one format over another continue to provoke debate. In reality, any color circle or color wheel, which presents a logically arranged sequence of pure hues, has merit.

Primary Colors

In traditional color theory, these are the 3 pigment colors that cannot be mixed or formed by any combination of other colors. All other colors are derived from these 3 hues



PRIMARY COLORS
Red, yellow and blue

© Copyright Virtual University of Pakistan

Secondary Colors

These are the colors formed by mixing the primary colors.



SECONDARY COLORS
Green, orange and purple

Tertiary colors

These are the colors formed by mixing one primary and one secondary color.



TERTIARY COLORS
Yellow-orange, red-orange, red-purple, blue-purple, blue-green and yellow-green.

Color Harmony

Harmony can be defined as a pleasing arrangement of parts, whether it be music, poetry, color, or even an ice cream sundae.

In visual experiences, harmony is something that is pleasing to the eye. It engages the viewer and it creates an inner sense of order, a balance in the visual experience. When something is not harmonious, it's either boring or chaotic. At one extreme is a visual experience that is so bland that the viewer is not engaged. The human brain will reject under-stimulating information. At the other extreme is a visual experience that is so overdone, so chaotic that the viewer can't stand to look at it. The human brain rejects what it cannot organize, what it cannot understand? The visual task requires that we present a logical structure. Color harmony delivers visual interest and a sense of order.

In summary, extreme unity leads to under-stimulation, extreme complexity leads to over-stimulation. Harmony is a dynamic equilibrium.

Some Formulas for Color Harmony

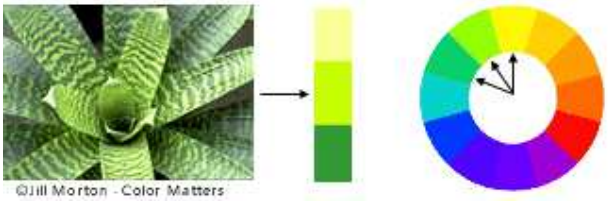
There are many theories for harmony. The following illustrations and descriptions present some basic formulas.

© Copyright Virtual University of Pakistan

Analogous colors

Analogous colors are any three colors, which are side by side on a 12 part color wheel, such as yellow-green, yellow, and yellow-orange. Usually one of the three colors predominates.

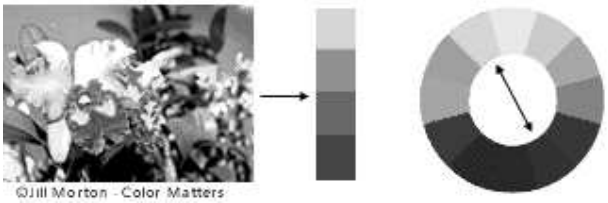
A color scheme based on analogous colors



Complementary colors

Complementary colors are any two colors, which are directly opposite each other, such as red and green and red-purple and yellow-green. In the illustration above, there are several variations of yellow-green in the leaves and several variations of red-purple in the orchid. These opposing colors create maximum contrast and maximum stability.

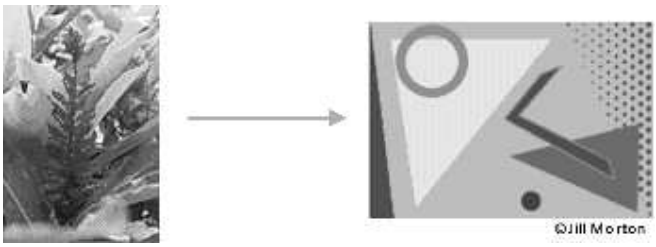
A color scheme based on complementary colors



Natural harmony

Nature provides a perfect departure point for color harmony. In the illustration above, red yellow and green create a harmonious design, regardless of whether this combination fits into a technical formula for color harmony.

A color scheme based on nature



Color Context

How color behaves in relation to other colors and shapes is a complex area of color theory. Compare the contrast effects of different color backgrounds for the same red square.

© Copyright Virtual University of Pakistan



Red appears more brilliant against a black background and somewhat duller against the white background. In contrast with orange, the red appears lifeless; in contrast with blue-green, it exhibits brilliance. Notice that the red square appears larger on black than on other background colors.

Different readings of the same color



As we age, the color of lens in eye changes. It becomes yellow and absorb shorter wavelengths so the colors with shorter wavelength will not be visible as we aged. So, do not use blue for text or small objects. As we age, the fluid between lens and retina absorbs more light due to which eye perceive lower level of brightness. Therefore older people need brighter colors.

Different wavelengths of light focused at different distances behind eye's lens this require constant refocusing which causes fatigue. So, be careful about color combinations. Pure (saturated) colors require more focusing then less pure. Therefore do not use saturated colors in User interface unless you really need something to stand out (danger sign).

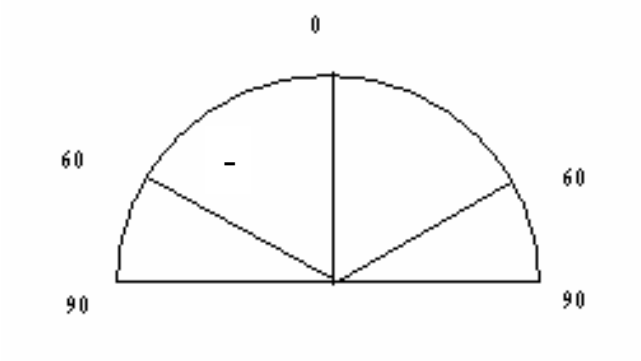
Guidelines

- Opponent colors go well together (red & green) or (yellow & blue)
- Pick non-adjacent colors on the hue circle
- Size of detectable changes in color varies. For example, it is hard to detect changes in reds, purples, & greens and easier to detect changes in yellows & blue-greens
- Older users need higher brightness levels to distinguish colors
- Hard to focus on edges created by color alone, therefore, use both brightness & color differences
- Avoid red & green in the periphery due to lack of RG cones there, as yellows & blues work in periphery
- Avoid pure blue for text, lines, & small shapes.

© Copyright Virtual University of Pakistan

- Blue makes a fine background color
- Avoid adjacent colors that differ only in blue
- Avoid single-color distinctions but mixtures of colors should differ in 2 or 3 colors., 2 colors shouldn't differ only by amount of red

Accurate color discrimination is at -+60 degree of straight head position.
Limit of color awareness is -+90 degree of straight head position



8.2 Stereopsis

Introduction

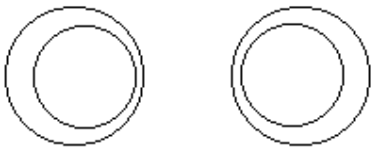
3D vision, *binocular vision* and *stereopsis* all mean the same thing: That remarkable power of the visual sense to give an immediate perception of depth on the basis of the difference in points of view of the two eyes. It exists in those animals with overlapping optical fields, acting as a range finder for objects within reach. There are many clues to depth, but stereopsis is the most reliable and overrides all others. The sensation can be excited by presenting a different, properly prepared, view to each eye. The pair of views is called a *stereopair* or *stereogram*, and many different ways have been devised to present them to the eye. The appearance of depth in miniature views has fascinated the public since the 1840's, and still appears now and then at the present time. There was a brief, but strong, revival in the 1990's with the invention of the autostereogram. Stereopsis also has technical applications, having been used in aerial photograph interpretation and the study of earth movements, where it makes small or slow changes visible.

The word stereopsis was coined from the Greek στερεος, solid or firm, and οψις, look or appearance. Since terms derived from Greek are often used in this field, it may be useful to have a brief discussion. Single and double vision are called *haplopia* and *diplopia*, respectively, from 'απλους (haplous) and διπλους (diplous), which mean "single" and "double". Haplopia is the happy case; with diplopia we are seeing double. The use of a Greek term removes the connotations that may attach to a common English word, and sounds much more scientific. Note that the "opia" part of these words refers to "appearance", and does not come from a word for "eye". The -s- has been dropped for euphony. Otherwise, the closest Greek to "opia" means a cheese from milk curdled with fig juice. "Ops", for that matter is more usually associated with cooked meat or evenings. In fact, words like "optic" come from οπτικος, meaning "thing seen", from the future οψσομαι of οραω, (horao) "to see", not from a reference to the eye. The Latin *oculus* does mean "eye" and is used in many technical terms, like *binocular*, which combines Greek and Latin.

Stereopsis is a term coined from Greek meaning "solid appearance," referring to the perception of depth through the combination of two slightly different images from each eye.

© Copyright Virtual University of Pakistan

Stereopsis



If you fuse this stereopair, you will see a frustum of a right cone in three dimensions.

A Simple Stereopair

Stereopsis is the direct sensing of the distance of an object by comparing the images received by the two eyes. This is possible only when the eyes of a creature look in the same direction, and have overlapping fields. The placing of the two eyes this way gives up the opportunity of a wide field of view obtained with eyes on the sides of the head. Predators find it best to have eyes in front, prey to have them on the sides. Stereopsis yields benefits for close work, such as fighting for cats and hand work for humans. Note that normal binocular vision is *single*, so that the two images have been *fused* by the brain. There is no evidence that the image resulting from the many simple eyes of an insect is not also fused in a similar way.

Visual perception makes use of a large number of distance clues to create its three-dimensional picture from the two-dimensional retinal images. Strong clues are the apparent sizes of objects of known size, overlapping and parallax, shadows and perspective. Weaker clues are atmospheric perspective (haze and scattering), speed of movement, and observed detail. The strongest clue of all, however, is stereopsis, which overrides all other evidence save touch itself. The convergence of the optic axes of the two eyes, and their distance accommodation, when fixated on an object, do not seem to be strong clues, though some have believed them to be. Although we have two eyes, we usually have only one visual world, which is a remarkable and important fact calling for explanation. Stereopsis gives a reliable distance clue as far away as 450 metres, Helmholtz estimated. The fineness of the comparison that must be made by the visual system is remarkable.

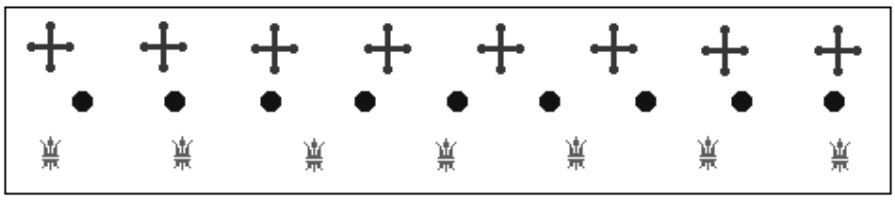
The interpretation of retinal images to produce stereopsis is entirely mental, and must be learned. When the images on the eyes are consistent with the observation of a single object, the two flat images *fuse* to form a vivid three-dimensional image. With practice, fusion can be achieved with two pictures side by side and the eyes voluntarily diverged so that each eye sees its picture straight ahead, though accommodated for the actual distance. Both the original pictures remain in view, but a third, fused, image appears before them when the concentration is diverted to it that appears strikingly solid. The brain regards this fused image as the real one, the others as mere ghosts. This skill is called *free fusion*, and requires considerable practice to acquire. In free fusion, both the convergence of the eyes, and their distance accommodation, are inconsistent with the actual location of the image, and must be overridden by stereopsis. It shows, incidentally, that convergence of the optic axes is not a strong depth clue. By the use of a *stereoscope*, one can achieve fusion without diverging the eyes, or focusing on a close object with the eyes so diverged, so no practice or skill is required. A stereoscope mainly changes the directions in which the two images are seen so that they can both be fixated by normally converged eyes. The two images are called a *stereo pair*.

When the images on the retinas are too different to be views of the same object, *rivalry* occurs, and either one image is favored and the other suppressed, or a patchwork of parts

© Copyright Virtual University of Pakistan

of the two images is seen. When everything corresponds except the illumination or colour, the fused image exhibits *lustre*.

The fundamentals of stereopsis were discovered by Charles Wheatstone in 1836, when stereopairs had to be created by drawing (this could be aided with the camera obscura, but was very difficult except for stick images). The methods of descriptive geometry can be used to create stereopairs. He designed the mirror stereoscope, which directs the view of the images into the eyes with plane mirrors and reflection at about 45 . David Brewster invented the prism stereoscope, which used prisms to deviate the light, which made a more compact and convenient apparatus. Lenses can also be used to decrease the viewing distance and make fixation easier. Photography was the natural way to create stereopairs. A stereopair can also be drawn in two colours with the views superimposed. When this *anaglyph* is viewed through coloured filters that present one image to each eye, fusion is easy. A similar method is to project the two images in orthogonal polarizations, and to view them through polarizing filters. Both of these methods have been used to project 3D films and transparencies before an audience. A small fraction of people, perhaps 4%, have defective stereopsis.



The pattern above demonstrates the stereoscopic *wallpaper* illusion, which was first discovered by H. Meyer in 1842, and also noted by Brewster. When viewed with the eyes parallel, a strong stereoscopic effect is seen. The green fleurs-de-lis are farthest away, the blue discs closest, and the red crosses at an intermediate distance. This is an *autosterogram*, a single figure that gives stereoscopic images to the two eyes. Since the figures in a line are identical, when the eyes are turned for free fusion, two different figures are assumed to be parallactic views of the same object. The eye finds it preferable to fuse the images rather than report double vision. It is easier to fuse this autostereogram than a normal stereopair, so it is good practice for developing the useful skill of free fusion.

The mind does not have to recognize the object in a stereopair for fusion to occur. The pattern can be random, but the stereopair must represent the same random pattern as seen from the different positions of the eyes (Julesz, 1960). Even more strikingly, a single apparently random pattern can be fused auto stereographically to give a three-dimensional image. No image is seen until fusion occurs. Each point on the image must be capable of interpretation as two different points of a stereopair. This random-dot auto stereogram was widely enjoyed in the 1980's. An auto stereogram requires free fusion, which must be learned in order to appreciate them. Many people found this difficult, so the auto stereogram was usually presented as a kind of puzzle.
Psychologists have argued about stereopsis for many years, but most of their musings are not worth repeating. A widely held theory was that the two retinas were somehow mapped point-by-point, and differing image positions with respect to this reference frame was interpreted stereoptically. It seems more likely to me that the images are compared by the visual sense for differences, than by their absolute locations on the retina. In the past, psychologists have preferred mechanical explanations, where the brain and retina

© Copyright Virtual University of Pakistan

are created with built-in specializations and functions, spatially localized, rather than regarding the organs as canvases, which the cognitive powers organize as necessary.

I have not discussed the broad and interesting field of optical illusions here, since they tell us nothing definite about the inner workings of the visual sense, only give examples of its operation, and also because the 'reasons' for them are controversial, and the arguments are not especially enlightening. Illusions are discussed at length in another article on this website. The oldest and most widely known illusion is the horizon illusion, in which the moon appears larger on the horizon than at the zenith. This illusion was known and discussed in antiquity, and is still the subject of much study. Its explanation is not known. For the application of the visual sense to investigation and appreciation of the world around us, Minaret's book is outstanding.

8.3 Reading

So far we have concentrated on the perception of images in general. However, the perception and processing of text is a special case that is important to interface design, which inevitably requires some textual display.

There are several stages in the reading process. First the visual pattern of the word on the page is perceived. It is then decoded with reference to an internal representation of language. The final stages of language processing include syntactic and semantic analysis and operate on phrases or sentences.

We are most interested with the first two stages of this process and how they influence interface design. During reading, the eye makes jerky movement called saccades followed by fixations. Perception occurs during the fixation periods, which account for approximately 94% of the time elapsed. The eye moves backwards over the text as well as forwards, in what are known as regressions. If the text is complex there will be more regressions.

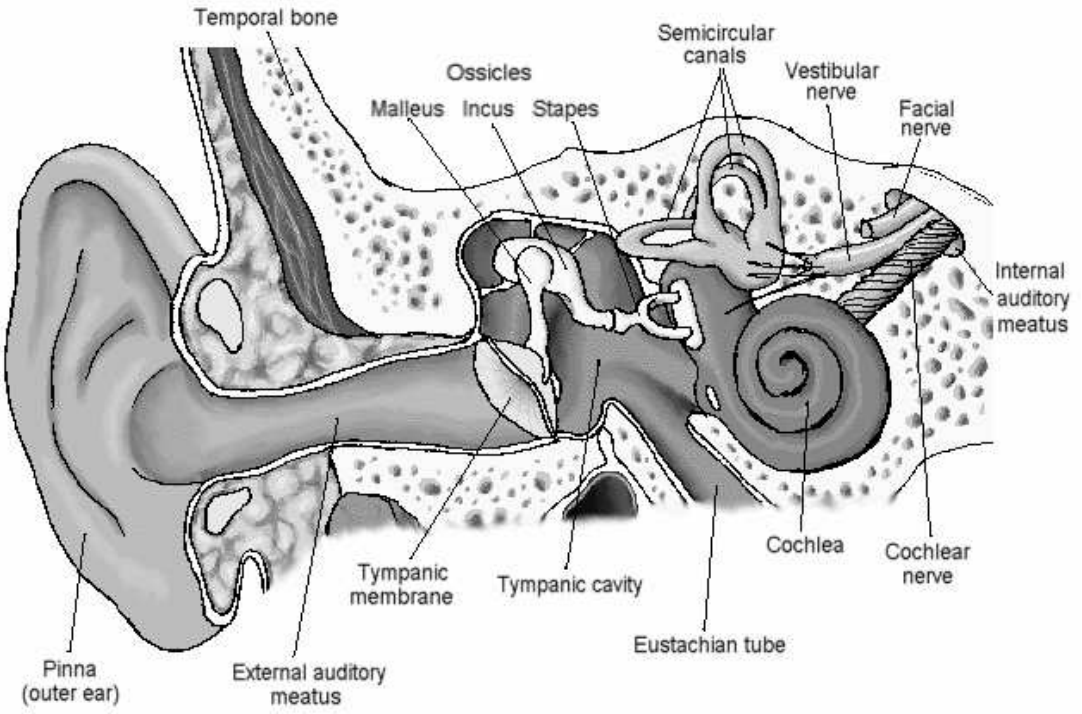
8.4 Hearing

The sense of hearing is often considered secondary to sight, but we tend to underestimate the amount of information that we receive through our ears.

The human ear

Hearing begins with vibrations in the air or sound waves. The ear receives these vibrations and transmits them, through various stages, to the auditory nerves. The ear comprises three sections commonly known as the outer ear, middle ear and inner ear.

© Copyright Virtual University of Pakistan



The outer ear is the visible part of the ear. It has two parts: the pinna, which is the structure that is attached to the sides of the head, and the auditory canal, along which sound waves are passed to the middle ear. The outer ear serves two purposes. First, it protects the sensitive middle ear from damage. The auditory canal contains wax, which prevents dust, dirt and over-inquisitive insects reaching the middle ear. It also maintains the middle ear at a constant temperature. Secondly, the pinna and auditory canal serve to amplify some sounds.

The middle ear is a small cavity connected to the outer ear by the tympanic membrane, or eardrum, and to the inner ear by the cochlea. Within the cavity are the ossicles, the smallest bones in the body. Sound waves pass along the auditory canal and vibrate the ear drum which in turn vibrates the ossicles, which transmit the vibrations to the cochlea, and so into the inner ear.

The waves are passed into the liquid-filled cochlea in the inner ear. Within the cochlea are delicate hair cells or cilia that bend because of the vibrations in the cochlean liquid and release a chemical transmitter, which causes impulses in the auditory nerve.

Processing sound

Sound has a number of characteristics, which we can differentiate.

Pitch

Pitch is the frequency of the sound. A low frequency produces a low pitch, a high frequency, a high pitch.

Loudness

Loudness is proportional to the amplitude of the sound; the frequency remains constant.

© Copyright Virtual University of Pakistan

Timber

Timber related to the type of the sound
Sounds may have the same pitch and loudness but be made by different instruments and so vary in timber.

Sound characteristics

Audible range is 20 Hz to 15 KHz. Human ear can distinguish between changes less than 1.5 Hz but less accurate at higher frequencies. Different frequencies trigger neuron activity causing nerve impulses. Auditory system filters sounds e.g., Cocktail Party Effect

8.5 Touch

The third sense is touch or haptic perception. Although this sense is ofttern viewed as less important than sight or hearing, imagine life without it. Touch provides us with vital information about our environment. It tells us when we touch something hot or cold, and can therefore act as a warning. It also provides us with feedback when we attempt to lfit and object.

Haptic perception involves sensors in the skin as well as the hand and arm. The movement that accompanies hands-on exploration involves different types of mechanoreceptors in the skin (involving deformation, thermo reception, and vibration of the skin), as well as receptors in the muscles, tendons, and joints involved in movement of the object (Verry, 1998). These different receptors contribute to a neural synthesis that interprets position, movement, and mechanical skin inputs. Druyan (1997) argues that this combination of kinesthetic and sensory perception creates particularly strong neural pathways in the brain.

Haptics vs. Visual

For the science learner, kinesthetic allows the individual to explore concepts related to location, range, speed, acceleration, tension, and friction. Haptics enables the learner to identify hardness, density, size, outline, shape, texture, oiliness, wetness, and dampness (involving both temperature and pressure sensations) (Druyan, 1997; Schiffman, 1976).

When haptics is compared to vision in the perception of objects, vision typically is superior with a number of important exceptions. Visual perception is rapid and more holistic—allowing the learner to take in a great deal of information at one time. Alternatively, haptics involves sensory exploration over time and space. If you give a student an object to observe and feel, the student can make much more rapid observations than if you only gave the student the object to feel without the benefit of sight. But of interest to science educators is the question of determining what a haptic experience adds to a visual experience. Researchers have shown that haptics is superior to vision in helping a learner detect properties of texture (roughness/ smoothness, hardness/ softness, wetness/ dryness, stickiness, and slipperiness) as well as micro spatial properties of pattern, compliance, elasticity, viscocity, and temperature (Lederman, 1983; Zangaladze, et al., 1999). Vision dominates when the goal is the perception of macro geometry (shape) but haptics is superior in the perception of micro geometry (texture) (Sathian et al., 1997; Verry, 1998). Haptics and vision together are superior to either alone for many learning contexts.

© Copyright Virtual University of Pakistan

While vision provides information about an object geometric feature, touch is unparalleled in its ability to extract information about materials. For a surgeon trying to decide where to begin excising a patch of cancerous tissue, it might be helpful to feel the texture and compliance, and not just rely on the shape**.**

Haptic Learning

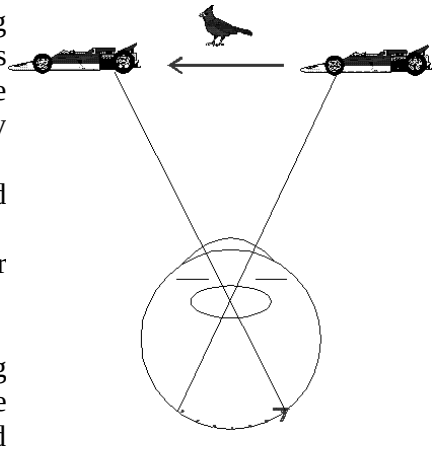
Haptic learning plays an important role in a number of different learning environments. Students with visual impairments depend on haptics for learning through the use of Braille as well as other strategies.

8.6 Movement

Before leaving this section on the human's input-output channels, we need to consider motor control and how the way we move affects our interaction with computers. A simple action such as hitting a button in response to a question involves a number of processing stages. The stimulus is received through the sensory receptors and transmitted to the brain. The question is processed and a valid response generated. The brain then tells the appropriate muscles to respond. Each of these stages takes time, which can be roughly divided into reaction time and movement time.

Movement time is dependent largely on the physical characteristics of the subjects: their age and fitness, for example. Reaction time varies according to the sensory channel through which the stimulus is received. A person can react to an auditory signal in approximately 150ms, to a visual signal in 200ms and to pain in 700ms.

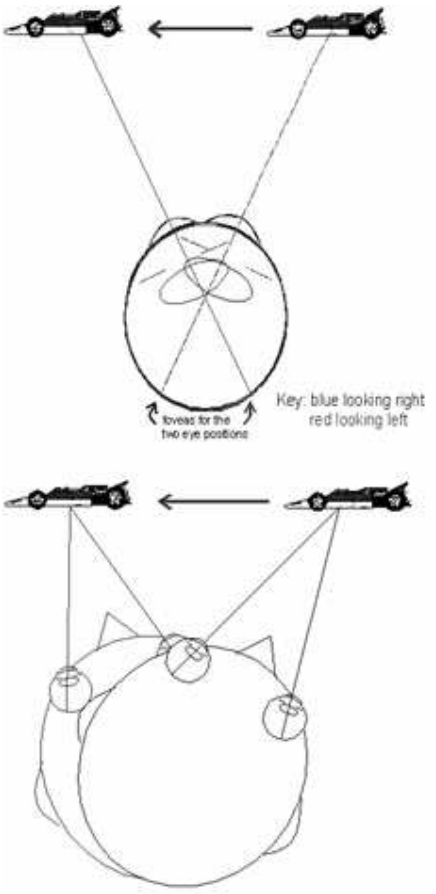
Movement perception



Assume that while you are staring at the bird, a racing car zooms by. The image of the car will travel across your retina as indicated by the dotted line with the arrow. This image movement will cause you to say that the car moves from your right to your left.
Now suppose you were looking at the car and followed its movement as it passes in front of you.
This time you are following the car by moving your eyes from right to left.

Just as before, your percept is that of the car moving from right to left. This is true even though the image remains on the fovea during the motion of the car and your eyes.
Third illustration shows that another way to follow the racing car is to keep the eyes steady and to move just the head. This causes the image to project to exactly the same retinal location at each instant (assuming you move your head at precisely the correct angular velocity) as the car moves from right to left.

© Copyright Virtual University of Pakistan



Once again, the percept is of the car moving from right to left. This percept will be the same as the two previous illustrations. How the brain distinguishes these different ways of following moving objects is the subject of much research. One more thing, although I have presented three distinct ways of following moving objects, these illustrations are gross simplifications. In point of fact, when we follow moving objects we use various combinations of head and eye movements.

The illustrations that, undoubtedly you have been looking at demonstrate that motion perception is very complex. Recall that we perceive motion if we hold our heads and eyes still as a moving object passes in front of us. If we decide to hold our heads still and let our eyes follow the object we still see it move. Finally, we could even decide to hold our eyes steady and move only our head to follow an object. The interesting thing is all three modes of viewing a moving object result in about the same perception.

So far we have been concerned with perceiving real movement. By real movement I mean that the physical stimulus is actually moving and we perceive it as moving. It is possible to perceive motion when the stimulus is not moving. An example is the motion after effect (MAE) demonstration that was loaned to me by Dr. Ben Bauer, Trent University.

Here is a demonstration you can observe for yourself. If you have the opportunity to view a waterfall, (e.g.. Niagara Falls) look at the falling water for about a minute and then allow your gaze to fall on any stationary object. A building would be excellent. If you do this, the texture of the building, perhaps even the windows will appear to move up.

© Copyright Virtual University of Pakistan

Waterfalls usually are not readily available. However, you can easily build your own MAE apparatus. Take a round paper plate. Draw a dozen or so heavy lines radiating out from the middle of the plate. Then with a pin attach the plate through its center to the eraser end of a pencil. Now spin the plate at a moderate speed. Don't spin it so fast that the lines become an indistinct blur. After viewing the spinning plate for about a minute stop it and continue to look at the radiating lines. What do you suppose you will see? If you see what most people notice the radiating lines, which are actually stationary, will appear to rotate in the direction opposite to that which you spun the plate originally. If that is way you saw you witnessed the MAE. It is useful to try this demonstration with the paper plate because it will convince you that there are no special tricks involved with the MAE demo I mentioned above.

The phenomenon of Motion after Effects (MAE) has been studied intensively by visual scientists for many years. One explanation of how the MAE works is the following. The visual system has motion detectors that, like most neurons, undergo spontaneous activity. You normally do not see motion when there is none because the spontaneous activity is in balance. However, when you viewed the downward motion of the black bars you adapted the motion detectors for motion in the downward direction. When the real motion stopped, the spontaneous activity was no longer in balance, the upward spontaneous activity being slightly stronger and thus the black bars appear to drift upward. The adaptation effect lasts for a short time, the motion detection system quickly becomes balanced again and the apparent movement stops.

Another example of motion being seen, when there is no physical motion, is the phi phenomenon. To those unacquainted with the field of vision research this phenomenon is probably unknown. However, all of you have seen it. The simplest demonstration of the phi phenomenon is to have two illuminated spots of light about 6 to 8 inches apart. When these lights alternately go on and off one usually see a single spot of light moving back and forth.

This principle is used in many movie marquees where one sees a pattern of lights moving around the display. In fact, there is no physical motion, only a series of lights going on and off. Then, of course there are the movies. Movies are a series of single frames presented in rapid succession. No one would doubt the perception of movement seen in the cinema. Yet, if you analyze the strips of film that yield these images all you would see is a series of frames each with a slightly different image. When they are rapidly projected on to the viewing screen motion is seen.
A similar technique is used with cartoons. The illustrator actually draws a series of pictures. When they are rapidly presented to the viewer motion of the cartoon characters is seen.

There are two other instances when movement is perceived. Have you ever sat in a train or bus station patiently waiting to get moving? Then all of a sudden, low and behold there you go. Or are you? You feel no vibration, something feels wrong. Then you notice that it is the vehicle (train or bus) right next to you that is moving and it just felt as if you were moving. This is called induced motion.

Finally, (and this is an experiment you can try at home) view a small very dim light in an otherwise completely dark room. Make sure that the light is in a fixed position and not

© Copyright Virtual University of Pakistan

moving. After sometime in the dark, the small light will appear to move somewhat randomly. This is called auto kinetic movement.

Here is another little experiment you can try. Look around your surroundings freely moving your eyes. As you move your eyes around are the stationary objects moving? Probably not. Now look at some object and with your finger rapidly press against your eyeball by pushing on your eyelid. (Don't push directly against the white (sclera) area). As you force your eye to move you will probably notice that whatever you are looking at starts to jump around. So you can see that it makes a difference whether you move your eyes normally or cause them to move in an unusual manner.

Electro physiologists are scientists who insert tiny electrode into the brain of experimental subjects. They have discovered that there are cortical neurons which are specialized for movement. In fact, these neurons often are so specialized that they will respond best when the motion is in a specific direction. E. Bruce Goldstein presents a neural model in his textbook, which shows how the early retinal neural processing could occur which results in a signal being sent to the brain which says that movement has occurred in a specific direction.

How to use MAE

Fixate the red square in the center of the diagram as the black bars move down.
When the black bars stop moving down, continue to fixate the red square and pay attention to the black bars. What if anything do the black bars appear to be doing? If they do not appear to do anything, try running the demonstration again by clicking on the refresh icon at the top of your screen. If the black bars appeared to be drifting upwards you witnessed the motion after effect. If you have a slow computer, a 486 machine or older, this demo may not work very well and you won't experience the MAE.

MAE (Motion After Effect): Perception of motion in the opposite direction after staring at a moving stimulus (e.g., spinning paper plate).

Cause of MAE: Adaptation of motion detectors in the visual system; imbalance in spontaneous neural activity creates the illusion of reverse motion.

Phi Phenomenon: Perceived motion from alternating lights (used in movie marquees and animations); no actual movement occurs.

Movies & Cartoons: Create motion illusion through rapid display of slightly varied static frames.

Induced Motion: Illusion of self-motion caused by movement of nearby objects (e.g., another train moving).

Autokinetic Effect: A stationary dim light appears to move in a dark room due to lack of visual reference.

Eye Movement Experiment: Pushing your eyeball causes stationary objects to appear to jump—shows how abnormal eye movement affects perception.

Neural Basis: Specialized cortical neurons detect motion in specific directions; early retinal processing sends motion signals to the brain.

© Copyright Virtual University of Pakistan

Lecture 9.

Cognitive Process - Part I

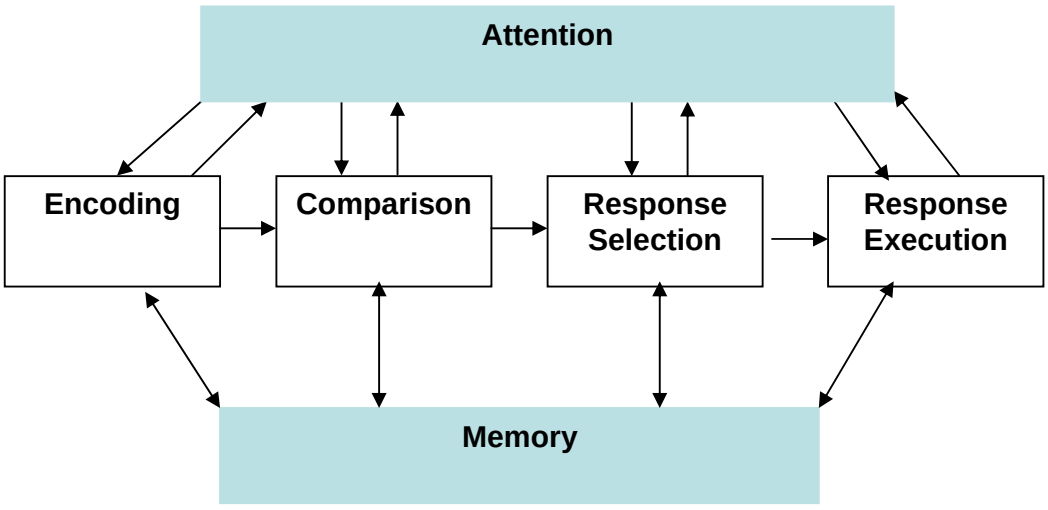
Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

Understand attention
Describe memory models
To day is first lecture on cognitive processes out series of two lectures on the same topic. In our previous lectures we have in detail talked about the cognitive psychology and cognitive frame works. Now is this lecture and in next coming lectures we will talk about a detail about the cognitive processes. As we have already discussed that cognition can be described in terms of specific kinds of processes. These include:

Attention
Memory
Perception and recognition
Learning
Reading, speaking and listening
Problem solving, planning, reasoning, decision-making.

Here in this lecture we will study the first two cognitive processes named attention and memory. The importance of these two you have seen in the Extended Human Processing model, studied in Lecture No. 6, as shown in figure



9.1 Attention

Attention is the process of selecting things to concentrate on, at a point in time, from the range of possibilities available.
A famous psychologist, Williams James says, " Everyone knows what attention is. It is the taking possession of mind, in clear and vivid form, of one out of what seem several

© Copyright Virtual University of Pakistan

simultaneously possible objects or trains of thought… it requires withdrawal from some things in order to deal effectively with others."
Attention involves our auditory and/or visual senses an example of auditory attention is waiting in the dentist's waiting room for our name to be called out to know when it is our time to go in. auditory attention is based on pitch, timber and intensity. An example of attention involving the visual senses in scanning the football results in a newspaper to attend to information about how our team has done. Visual attention is based on color and location.
Attention allows us to focus on information that is relevant to what we are doing. The extent to which this process is easy or difficult depends on
Whether we have clear goals and
Whether the information we need is salient in the environment.

Our goals
If we know exactly what we want to find out, we try to match this with the information that is available. For example, if we have just landed at an airport after a long flight and want to find out who has won the World Cup, we might scan the headlines at the newspaper stand, check the web, call a friend, or ask someone in the street.
When we are not sure exactly what we are looking for we may browse through information, allowing it to guide our attention to interesting or salient items. For example, when we go to restaurant we may have the general goal of eating a meal but only a vague idea of what we want to eat. We peruse the menu to find things that whet our appetite, letting our attention be drawn to the imaginative descriptions of various dishes. After scanning through the possibilities and imagining what each dish might be like (plus taking into account other factors, such as cost, who we are with, what the specials are, what the waiter recommends, whether we want a two-or- three-course meal, and so on.), we may then make a decision.

Information presentation
The way information is displayed can also greatly influence how easy or difficult it is to attend to appropriate pieces of information. Look at the figure below, two different ways of structuring the same information at the interface: one makes it much easier to find information than the other. Look at the top screen and (i) find the price for a double room at the Holiday Inn in Lahore: (ii) find the phone number of the Sheraton in the Karachi. Then look at the bottom screen and (i) find the price of for a double room at the Pearl Continental in Faisalabad; (ii) find the phone number of the Holiday Inn in the Islamabad. Which took longer to do? Experiments showed that the two screens produced quite different results: it took an average of 3.2 seconds to search the top screen and 5.5 seconds to find the same kind of information in the bottom screen. Why is this so, considering that the both displays have the same density of information? The primary reason is the way the characters are grouped in the display; in the top they are grouped into vertical categories of information that have columns of space between them. In the bottom screen the information is bunched up together, making it much harder to search through.

© Copyright Virtual University of Pakistan

City	Hotel	Area Code	Phone	Rates Single	Rates Double
Lahore	Holiday Inn	042	6300634	2000	4000
Lahore	Sheraton	042	5456322	1800	3500
Lahore	Perl Continental	042	4565654	2500	5000
Karachi	Holiday Inn	021	1645656	2000	4000
Karachi	Sheraton	021	6545646	1800	3500
Karachi	Perl Continental	021	9545656	2500	5000

```
Islamabad: Holiday Inn
(051) 4544556 S:2000 D:4000
Islamabad: Sheraton
(051) 2135466 S:1500 D:3500
Islamabad: Pearl Continental
(051) 6565322 S:3000 D:6000
Faisalabad: Holiday Inn
(041) 4544556 S:2000 D:4000
Faisalabad: Sheraton
(041) 2135466 S:1500 D:3500
Faisalabad: Pearl Continental
(041) 6565322 S:3000 D:6000
```

Focused Attention: The ability to concentrate on one specific stimulus while ignoring others.

Divided Attention: The ability to attend to and process multiple stimuli or tasks at the same time.

Voluntary Attention: Attention that is consciously and intentionally directed toward something.

Models of Attention

There are two models of attention:

- Focused attention
- Divided attention

Focused Attention

Our ability to attend to one event from what amounts to a mass of competing stimuli in the environment have been psychologically termed as focused attention. The streams of information we choose to attend to will tend to be relevant to the activities and intentions that we have at that time. For example, when engaged in a conversation it is usual to attend to what the other person is saying. If something catches our eye in the periphery to our vision, for example, another person we want to talk to suddenly appear, we may divert our attention to what she is doing. We may then get distracted from the conversation we are having and as a consequence have to ask the person we are conversing with to repeat them. On the other hand, we may be skilled at carrying on the conversation while intermittently observing what the person we want to talk to is doing.

Divided Attention

As we said, we may be skilled at carrying on the conversation while intermittently observing what the person we want to talk to is doing. When we attempt to attend to mire than one thing at a time, as in the above example, it is called divided attention. Another example that is often used to illustrate this intentional phenomenon is being able to drive while holding a conversation with a passenger.

Voluntary attention

A further property of attention is that can be voluntary, as when we make a conscious effort to change our attention.

© Copyright Virtual University of Pakistan

Involuntary Attention
Attention that is automatically captured by sudden or prominent stimuli without conscious control.

Involuntary attention
Attention may also be involuntary, as when the salient characteristics of the competing stimuli grab our attention. An everyday example of an involuntary act is being distracted from working when we can hear music or voices in the next room.
Another thing is that frequent actions become automatic actions, that is, they do not need any conscious attention and they require no conscious decisions.

Focusing attention at the interface
What is the significance of attention for HCI? How can an understanding of intentional phenomena be usefully applied to interface design? Clearly, the manner in which we deploy our attention has a tremendous bearing on how effectively we can interact with a system. If we know that people are distracted, often involuntarily, how is it possible to get their attention again without allowing them to miss the 'window of opportunity'? Moreover, how can we focus people's attention on what they need to be looking at or listening to for any given stage of task? How can we guide their attention to the relevant information on display?

Structuring Information
One way in which interfaces can be designed to help users find the information they need is to structure the interface so that it is easy to navigate through. Firstly. This requires presenting not too much information and not too little o a screen, as in both cases the user will have to spend considerable time scanning through either a cluttered screen or numerous screens of information. Secondly, instead of arbitrarily presenting data I the screen it should be grouped and ordered into meaningful parts capitalizing o the perceptual laws of grouping, information can be meaningfully structured so that it is easier to perceive and able to guide attention readily to the appropriate information.
Help user to.

Attend his/her task not the interface.
Decide what to focus on, based on their tasks, interest, etc.
Stay focused, do not provide unnecessary distractions.
Structure his/her task, e.g. help
Create distraction, when really necessary!
Use alerts (only) when appropriate!

Some other considerations are as under:
Make information salient when it needs attending to
Use techniques that make things stand out like colour, ordering, spacing, underlining, sequencing and animation
Avoid cluttering the interface - follow the google.com example of crisp, simple design
Avoid using too much because the software allows it

9.2 Memory

Indeed, much of our everyday activities rely on memory. As well as storing all our factual knowledge, our memory contains our knowledge of actions or procedures. It allows us to repeat actions, to use language, and to use new information received wia our senses. It also gives us our sense of identity, by preserving information from our past experiences. If want to understand the working of our memory, it is necessary to understand the structure of memory. Let us look at a memory model.

© Copyright Virtual University of Pakistan

Memory Model

It is generally agreed that there are three types of memory or memory functions sensory buffers, short-term memory or working memory and long-term memory. It is also called the multi-store model of memory. The main characteristics of the multi-store model of memory are the various types of memory stores. These are:

Sensory store modality-specific, hold information for a very brief period of time (a few tenth of a second),

Short-term memory store holds limited information for a short period of time (a few seconds),

Permanent long-term memory store hold information indefinitely.

Lets us have a detailed look of this model.

Sensory Memory: A very short-term memory that briefly holds information from our senses like sight, sound, and touch.

Sensory memory

The sensory memories act as buffer for stimuli received through the senses. A sensory memory exists for each sensory channel: iconic memory for visual stimuli, echoic memory for aural stimuli and haptic memory for touch. These memories are constantly overwritten by new information coming in on these channels.

We can demonstrate the existence of iconic memory by moving a finger in front of the eye. Can you see it in more than one place at once? This indicates a persistence of the image after the stimulus has been removed. A similar effect is noticed most vividly at firework displays where moving sparklers leave a persistent image. Information remains in iconic memory very briefly, in the order of 0.5 seconds.

Similarly, the existence of echoic memory is evidenced by our ability to ascertain the direction from which a sound originates. This is due to information being received by both ears. However, since this information is received at different times, we must store the stimulus in the meantime. Echoic memory allows brief playback of information. Have you ever had someone ask you a question when you are reading? You ask them to repeat the question only to realise that you know what was asked after all. This experience, too, is evidence of the existence of echoic memory. Information is passed from sensory memory into short-term memory by attention, thereby filtering the stimuli to only those, which are of interest at a given time. Attention is the concentration of the mind on one out of a number of competing stimuli or thoughts. It is clear that we are able to focus our attention selectively, choosing to attend to one thing rather than another. This is due to the limited capacity of our sensory and mental processes. If we did not selectively attend to the stimuli coming into our senses, we would be overloaded. We can choose which stimuli to attend to, and the choice is governed to an extent by our arousal, our level of interest or need. This explains the cocktail party phenomenon. According to cocktail party effect we can attend to one conversation over the background noise, but we may choose to switch our attention to a conversation across the room if we hear our name mentioned. Information received by sensory memories is quickly passed into a more permanent memory store, or overwritten and lost.

© Copyright Virtual University of Pakistan

Short-Term Memory: A temporary memory system used to hold and process small amounts of information for brief periods, like during mental calculations or reading.

Short term memory

Short-term memory or working memory acts as a scratch pad for temporary recall of information. It is use to store information which is only required fleetingly. For example, calculate the multiplication 35 * 6 in your head. The chances are that you will have done this calculation in staged, perhaps 5 ***** 6 and then 3 ***** 6 and added the results. To perform calculations such as this we need to store the intermediate stages for use later. Or consider reading. In order to comprehend this sentence you need to hold in your mind the beginning of the sentence as you read the rest. Both of these tasks use short-term memory.

Short-term memory can be accessed rapidly, in the order of 70ms. However, it also decays rapidly, meaning that information can only be held there temporarily, in the order of 200ms.

Short-term memory also has a limited capacity. There are two basic methods for measuring memory capacity. The first involves determining the length of a sequence, which can be remembered in order. The second allows items to be freely recalled in any order. Using the first measure, the average person can remember 7±2 digits. This was established in experiments by Miller. Try it look at the following number sequence:

54988319814237

Now write down as much of the sequence as you can remember. Did you get it all right? If not, how many digits could you remember? If remembered between five and nine digits your digits your digit span is average.

Now try the following sequence:

22 55 36 8998 30

Did you recall that more easily? Here the digits are grouped or chunked. A generalization of the 7±2 rule is that we can remember 7±2 chunks of information. Therefore chunking information can increase the short-term memory capacity. The limited capacity of short-term memory produces a subconscious desire to create chunks, and so optimise the use of the memory. The successful formation of a chunk is known as closure. This process can be generalized to account for the desire to complete or close tasks held in short-term memory. If a subject fails to do this or is prevented from doing so by interference, the subject is liable to lose trick of what she is doing and make consequent errors.

Recency Effect: The tendency to remember the last items in a list better because they are still in short-term memory at the time of recall.

Recency effect

In experiments where subjects were able to recall works freely, evidence shows that recall of the last words presented is better than recall of those in the middle. This is known as recency effect. Recency effect can be defined as: ‘better recall for items at the end of the list because these items are still active in STM (and possibly SM) at time of recall’.

However, if the subject is asked to perform another task between presentation and recall the recency effect is eliminated. The recall of other words is unaffected. This suggests that short-term memory recall is damaged by interference of other information.

Primacy effect

You remember the first things better because you had more time to practice or repeat them.

‘Better recall for items at the beginning of the list (because these items have been rehearsed more frequently than other items and thus have a greater chance of being placed in LTM).’

Long-Term Memory: A large, long-lasting memory store where we keep everything we know, like facts, experiences, and skills, with slow forgetting and almost unlimited capacity.

Long Term Memory

If short-term memory is our working memory or ‘scratch-pad’, long-term memory is our main resource. Here we store factual information, experiential knowledge, and procedural

© Copyright Virtual University of Pakistan

rules of behavior- in fact, everything that we know. It differs from short-term memory in a number of significant ways. First, it has a huge, if not unlimited, capacity. Secondly, it has a relatively slow access time of approximately a tenth of a second. Thirdly, forgetting occurs more slowly in long-term memory, if at all.
Long-term memory is intended for the long-term storage of information. Information is placed there from working memory through rehearsal. Unlike working memory there is little decay: long-term recall after minutes is the same as that after hours or days.

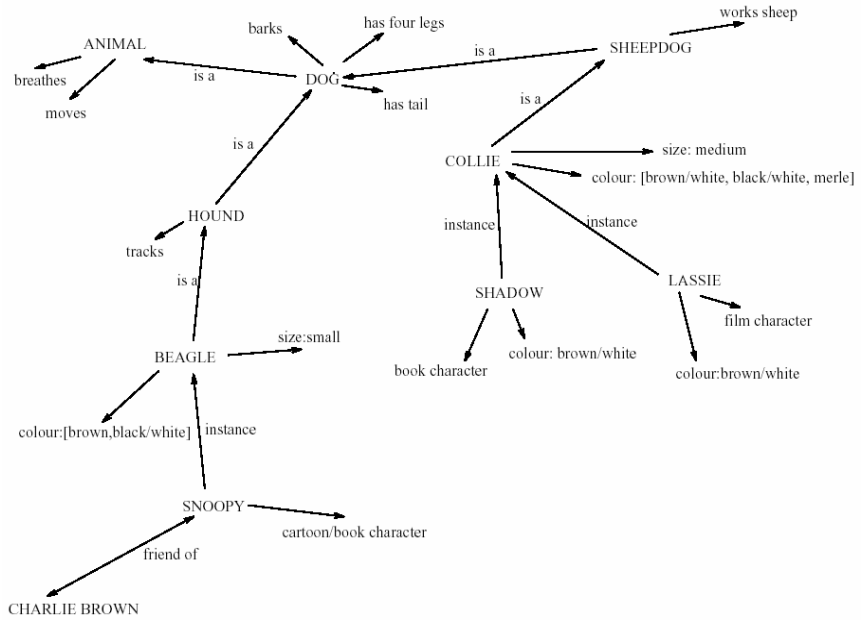
Long-term memory structure
There are two types of long-term memory: episodic memory and semantic memory.

Episodic memory
Episodic memory represents our memory of events and experiences in a serial form. It is from this memory that we can reconstruct the actual events that took place at a given period of our lives.

Semantic memory
Semantic memory is structured record of facts, concepts and skills that we have acquired. The information in semantic memory is derived from that in our episodic memory, such that we can learn new facts or concepts from our experience.
Semantic memory is structured in some way to allow access to information, representation of relationships between pieces of information, and inference. One model for the way in which semantic memory is structured is as a network. Items are associated to each other in classes, and may inherit attributes from parent classes. This model is known as a semantic network. As an example, knowledge about dogs may be stored in a network such as that shown in figure.
Specific breed attributes may be stored with each given breed; yet general dog information is stored at a higher level. This allows us to generalize about specific cases. For instance, we may not have been told that the sheepdog Shadow has four legs and a tail, but we can infer this information from our general knowledge about sheepdogs and dogs in general. Note also that there are connections within the network which link into other domains of knowledge, for example cartoon characters

A number of other memory structures have been proposed to explain how we represent and store different types of knowledge. Each of these represents a different aspect of knowledge and, as such, the models can be viewed as complementary rather than mutually exclusive. Semantic networks represent the associations and relationship between single items in memory. However, they do not allow us to model the representation of more complex objects or events, which are perhaps composed of a number of items of activities. Structured representations such as frames and scripts organize information into data structures. Slots in these structures allow attribute values to be added. Frame slots may contain default, fixed or variable information. A frame is instantiated when the slots are filled with appropriate values. Frame and scripts can be linked together in networks to represent hierarchical structured knowledge.

© Copyright Virtual University of Pakistan



DOG

Fixed
legs: 4

Default
diet: carniverous
sound: bark

Variable
size:

COLLIE

Fixed
breed of: DOG
type: sheepdog

Default
size: 65 cm

Variable
colour

Script for a visit to the vet

Entry conditions: *dog ill*
vet open
owner has money

Result: *dog better*
owner poorer
vet richer

Props: *examination table*
medicine
instruments

Roles: *vet examines*
diagnoses
treats
owner brings dog in
pays
takes dog out

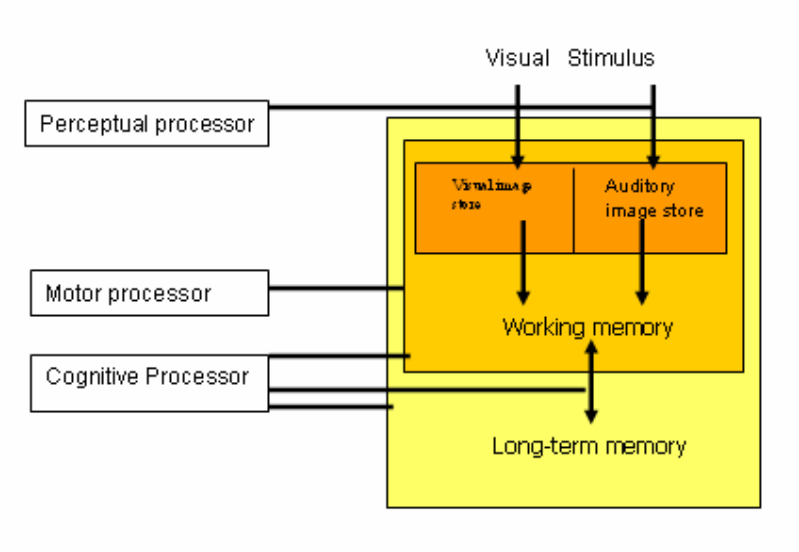
Scenes: *arriving at reception*
waiting in room
examination
paying

Tracks: *dog needs medicine*
dog needs operation

© Copyright Virtual University of Pakistan

9.3 **Revised Memory Model**

According to revised memory model working memory is a subset of LTM.
Some other characteristics are as under:
Items are semantically linked.
Items in working memory are activated.
Activation is supplied from other linked chunks and from sensory input.
As we know human processor consists of three interacting systems: the perceptual system, the motor system and the cognitive system. Each has its own memory and process as shown in figure. Similar to the notions of human information processing, human performance is viewed as a series of processing stages, whereby the different processors and memories are organized in a particular way.



© Copyright Virtual University of Pakistan

Lecture 10.

Cognitive Processes - Part II

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand learning
- Discuss planning, reasoning, decision making
- Understand problem solving

Today is part second of our two parts series lecture on Cognitive Process. As we have earlier seen that cognition involves following processes:

- Attention
- Memory
- Perception and recognition
- Learning
- Reading, speaking and listening
- Problem solving, planning, reasoning, decision-making.

Today we will learn about learning and thinking. Let us first look at learning.

10.1 Learning

Learning can be considered in two terms:

- Procedural
- Declarative

Procedural

According to procedural learning we come to any object with questions like how to use it? How to do something? For example, how to use a computer-based application?

Declarative

According to declarative learning we try to find the facts about something. For example, using a computer-based application to understand a given topic.

Jack Carroll and his colleagues have written extensively about how to design interfaces to help learners develop computer-based skills. A main observation is that people find it very hard to learn by following sets of instructions in a manual. For example, when people encounter a computer for the first time their most common reaction is one of fear

© Copyright Virtual University of Pakistan

and trepidation. In contrast, when we sit behind the steering wheel of a car for the first time most of us are highly motivated and very excited with the prospect of learning to drive. Why, then m is there such a discrepancy between our attitudes to learning these different skills? One of the main differences between the two domains is the way they are taught. At the end of the first driving lesson, a pupil will have usually learned how to drive through actually doing. This includes performing a number of complex tasks such as clutch control, gear changing, learning to use the controls and knowing what they are. Furthermore, the instructors are keen to let their pupils try thing out and get started. Verbal instruction initially is kept to minimum and usually interjected only when necessary. In contrast, someone who sits in front of a computer system for the first time may only have a very large manual, which may be difficult to understand and poorly presented. Often training and reference materials are written as a series of ordered explanations together with step by step exercises, which may cause the learner to feel overloaded with information or frustrated at not being able to find information that she wants. One of the main developing usable training materials and helps facilities. There is general assumption that having read something in the manual users can immediately match it to what is happening at the interface and respond accordingly. But as you may have experienced, trying to put into action even simple descriptions can sometimes be difficult.

Experienced users also appear to be reluctant to learn new methods and operations from manuals. When new situations arise that could be handled more effectively by new procedures, experienced users are more likely to continue to use the procedures they already know rather than try to follow the advanced procedures outlined in a manual, even if the former course takes much longer and is less effective.

So, people prefer to learn through doing. GUI and direct manipulation interface are good environments for supporting this kind of learning by supporting exploratory interaction and importantly allowing users to ‘undo’ their actions, i.e., return to a previous state if they make a mistake by clicking on the wrong option.

Carroll has also suggested that another way of helping learners is by using a ‘training wheels’ approach. This involves restricting the possible functions that can be carried out by a novice to the basics and then extending these as the novice becomes more experienced. The underlying rationale is to make initial learning more tractable, helping the learner focus on simple operations before moving on to more complex ones.

There have also been numerous attempts to harness the capabilities of different technologies, such as web-based, multimedia, and virtual reality, is that they provide alternative ways of representing and interacting with information that are not possible with traditional technologies. In so doing, they have the potential of offering learners the ability to explore ideas and concepts different ways.

People often have problems learning the difficult stuff---by this we mean mathematical formulae, notations, laws of physics, and other abstract concepts. One of the main reasons is that they find it difficult to relate their concrete experiences of the physical world with these higher-level abstractions. Research has shown, however, that it is possible to facilitate this kind of learning through the use of interactive multimedia.

© Copyright Virtual University of Pakistan

Dynalinking

The process of linking and manipulating multimedia representations at the interface is called dynalinking. It is helpful in learning. An example where dynalinking have been found beneficial is in helping children and students learn ecological concepts. During experiment a simple ecosystem of a pond was built using multimedia. The concrete simulation showed various organisms swimming and moving around and occasionally an event where one would eat another. When an organism was clicked on, it would say what it was and what it ate.

The simulation was dynalinked with other abstract representations of the pond ecosystem. One of these was a food web diagram. People were encouraged to interact with the interlinked diagrams in various ways and to observe what happened in the concrete simulation when something was changed n the diagram and vice versa.
Dynalinking is a powerful form of interaction and can be used in a range of domains to explicitly show relationships among multiple dimensions, especially when the information to be understood or learned is complex.

10.2 Reading, Speaking and Listening

These three forms of language processing have both similar and different properties. One similarity is that the meaning of sentences or phrases is the same regardless of the mode in which it is conveyed. For example, the sentence "Computer are a wonderful invention" essentially has the same meaning whether one reads it, speaks it, or hears it. However, the ease with which people can read, listen, or speak differs depending on the person, task, and context. For example, many people find listening much easier than reading. Specific differences between the three modes include:

- Written language is permanent while listening is transient. It is possible to reread information if not understood the first time round. This is not possible with spoken information that is being broadcast.
- Reading can be quicker than speaking or listening, as written text can be rapidly scanned in ways not possible when listening to serially presented spoken works.
- Listening require less cognitive effort than reading or speaking. Children, especially, often prefer to listen to narratives provided in multimedia or web-based learning material than to read the equivalent text online.
- Written language tends to be grammatical while spoken language is often ungrammatical. For example, people often start and stop in mid-sentence, letting someone also start speaking.
- There are marked differences between people in their ability to use language. Some people prefer reading to listening, while others prefer listening. Likewise, some people prefer speaking to writing and vice versa.
- Dyslexics have difficulties understanding and recognizing written words, making it hard for them to write grammatical sentences and spell correctly.
- People who are hard of hearing or hart of seeing are also restricted in the way they can process language.

Dyslexics un logon ko kehte hain jo dyslexia ka shikar hote hain — yani unhain pardhne, likhne, ya lafzon ko samajhne mein mushkil hoti hai, lekin unki intelligence normal hoti hai.

© Copyright Virtual University of Pakistan

Incorporating Language processing in applications

Many applications have been developed either to capitalize on people's reading writing and listening skills, or to support or replace them where they lack or have difficulty with them. These include:

- Interactive books and web-based material that help people to read or learning foreign languages.
- Speech-recognition systems that allow users to provide instructions via spoken commands.
- Speech-output systems that use artificially generated speech
- Natural-language systems that enable users to type in questions and give text-based responses.
- Cognitive aids that help people who find it difficult to read, write, and speak. A number of special interfaces have been developed for people who have problems with reading, writing, and speaking.
- Various input and output devices that allow people with various disabilities to have access to the web and use word processors and other software packages.

Design Implications

- Keep the length of speech-based menus and instructions to a minimum. Research has shown that people find it hard to follow spoken menu with more than three or four options. Likewise, they are bad at remembering sets of instructions and directions that have more than a few parts.
- Accentuate the intonation of artificially generated speech voices, as they are harder to understand than human voices.
- Provide opportunities for making text large on a screen, without affecting the formatting, for people who find it hard to read small text.

10.3 Problem Solving, Planning, Reasoning and Decision-making

Problem solving, planning, reasoning and decision-making are all cognitive processes involving reflective cognition. They include thinking about what to do, what the options are, and what the consequences might be of carrying out a given action. They often involve conscious processing (being aware of what one is thinking about), discussion with others, and the use of various kinds of artifacts, (e.g., maps, books, and pen and paper). For example, when planning the best route to get somewhere, say a foreign city, we may ask others use a map, get instructions from the web, or a combination of these.
Reasoning also involves working through different scenarios and deciding which is the best option or solution to a given problem. In the route-planning activity we may be aware of alternative routes and reason through the advantages and disadvantages of each route before deciding on the best one. Many family arguments have come about because one member thinks he or she knows the best route while another thinks otherwise.

Comparing different sources of information is also common practice when seeking information on the web. For example, just as people will phone around for a range of

© Copyright Virtual University of Pakistan

quotes, so too, will they use different search engines to find sites that give the best deal or best information. If people have knowledge of the pros and cons of different search engines, they may also select different ones for different kinds of queries. For example, a student may use a more academically oriented one when looking for information for writing an essay, and a more commercially based one when trying to find out what's happening in town.

The extent to which people engage in the various forms of reflective cognition depends on their level of experience with a domain; applications about what to do using other knowledge about similar situations. They tend to act by trial and error, exploring and experimenting with ways of doing things. As a result they may start off being slow, making errors and generally being inefficient. They may also act irrationally, following their superstitions and not thinking ahead to the consequences of their actions. In contrast experts have much more knowledge and experience and are able to select optimal strategies for carrying out their tasks. They are likely to able to think ahead more, considering what the consequences might be of opting for a particular move or solution.

Reasoning

Reasoning is the process by which we use the knowledge we have to draw conclusions or infer something new about the domain of interest. There are a number of different types of reasoning:

- Deductive reasoning
- Inductive reasoning
- Abdicative reasoning

Deductive reasoning

Deductive reasoning derives the logically necessary conclusion from the given premises. For example,

It is Friday then she will go to work
It is Friday
Therefore she will go to work

It is important to note that this is the logical conclusion from the premises; it does not necessarily have to correspond to our notion of truth. So, for example,

If it is raining then the ground is dry
It is raining
Therefore the ground is dry.

Is a perfectly valid deduction, even though it conflicts with our knowledge of what is true in the world?

Inductive reasoning

Induction is generalizing from cases we have seen to infer information about cases we have not seen. For example, if every elephant we have ever seen has a trunk, we infer that all elephants have trunks. Of course, this inference is unreliable and cannot be proved to be true; it can only be proved to be false. We can disprove the inference simply by producing an elephant without a trunk. However, we can never prove it true because, no matter how many elephants with trunks we have seen or are known to exist, the next one

© Copyright Virtual University of Pakistan

Deductive Reasoning

Definition: Drawing a logically necessary conclusion from given premises.
Example:

Premise: It is Friday, then she will go to work.
Premise: It is Friday.
Conclusion: Therefore, she will go to work.

Inductive Reasoning

Definition: Generalizing from specific cases to make broader conclusions, though it can't be proven true.

Example: If every elephant we've seen has a trunk, we infer that all elephants have trunks.

Abductive Reasoning

Definition: Reasoning from an observed fact to the most likely cause or explanation.

Example: If Sam is driving too fast, we may infer she has been drinking, though there could be another reason.

we see may be trunkless. The best that we can do is gather evidence to support our inductive inference.

In spite of its unreliability, induction is a useful process, which we use constantly in learning about our environment. We can never see all the elephants that have ever lived or will ever live, but we have certain knowledge about elephants, which we are prepared to trust for all practical purposes, which has largely been inferred by induction. Even if we saw an elephant without a trunk, we would be unlikely to move from our position that 'All elephants have trunk', since we are better at using positive than negative evidence.

Abdicative reasoning

The third type of reasoning is abduction. Abduction reasons from a fact to the action or state that caused it. This is the method we use to derive explanations for the events we observe. For example, suppose we know that Sam always drives too fast when she has been drinking. If we see Sam driving too fast we may infer that she has been drinking. Of course, this too is unreliable since there may be another reason why she is driving fast: she may have been called to an emergency, for example.

In spite of its unreliability, it is clear that people do infer explanations in this way and hold onto them until they have evidence to support an alternative theory or explanation. This can lead to problems in using interactive systems. If an event always follows an action, the user will infer that the event is caused by the action unless evidence to the contrary is made available. If, in fact, the event and the action are unrelated, confusion and even error often result.

Problem solving

If reasoning is a means of inferring new information from what is already known, problem solving is the process of finding a solution to an unfamiliar task, using the knowledge we have. Human problem solving is characterized by the ability to adapt the information we have to deal with new situations. However, often solutions seen to be original and creative. There are a number of different views of how people solve problems. Te earliest, dating back to the first half of the twentieth century is the Gestalt view that problem solving involves both reuse of knowledge and insight. This has been largely superseded but the questions it was trying to address remain and its influence can

be seen in later research. A second major theory, proposed in the 1970s by Newell and Simon, was the problem space theory, which takes the view that the mind is a likited information processor. Later variations on this drew on the earlier thory and attempted to reinterpret Gestalt theory in terms of information-processing theories. Let us look at these theories.

Gestalt theory says problem solving isn't just trial and error—it can also involve insight and restructuring of the problem. While we sometimes use past experience (reproductive), real problem solving often requires a fresh view (productive). However, the theory lacks clear explanation of how or when insight happens.

Gestalt theory

Gestalt psychologists were answering the claim, made by behaviorists, that problem solving is a matter of reproducing known responses or trial and error. This explanation was considered by the Gestalt school to be insufficient to account for human problem-solving behavior. Instead, they claimed, problem solving is both productive and reproductive. Reproductive problem solving draws on previous experience as the behaviorist claimed, but productive problem solving involves insight and restructuring of the problem. Indeed, reproductive problem solving could be hindrance to finding a

© Copyright Virtual University of Pakistan

solution, since a person may 'fixate' on the known aspects of the problem and so be unable to see novel interpretations that might lead to a solution.
Although Gestalt theory is attractive in terms of its description of human problem solving, it does not provide sufficient evidence or structure to support its theories. It does not explain when restructuring occurs or what insight is, for example.

Problem space theory

Problem space theory says that solving a problem is like moving from a starting point (initial state) to a goal using steps (operators). Since the number of possible steps can be huge, we use smart shortcuts (heuristics) like means-ends analysis, where we compare where we are with where we want to go and take actions to reduce the difference. Our memory and thinking speed can affect how well we do this.

Newell and Simon proposed that problem solving centers on the problem space. The problem space comprises problem states, and problem solving involves generating these states using legal state transition operators. The problem has an initial state and a goal state and people use the operator to move from the former to the latter. Such problem spaces may be huge, and so heuristics are employed to select appropriate operators to reach the goal. One such heuristic is means-ends analysis. In means-ends analysis the initial state is compared with the goal state and an operator chosen to reduce the difference between the two. For example, imagine you are recognizing your office and you want to move your desk from the north wall of the room to the window. Your initial state is that the desk is at the north wall. The goal state is that the desk is by the window. The main difference between these two is the location of your desk. You have a number of operators, which you can apply to moving things: you can carry them or push them or drag them, etc. however, you know that to carry something it must be light and that your desk is heavy. You therefore have a new sub-goal: to make the desk light. Your operators for this may involve removing drawers, and so on.

An important feature of Newell and Simon's model is that it operates within the constraints of the human processing system, and so searching the problem space is limited by capacity of short-term memory, and the speed at which information can be retrieved. Within the problem space framework, experience allows us to solve problems more easily since we can structure the problem space appropriately and choose operators efficiently.

Analogy in problem solving

A third element of problem solving is the use of analogy. Here we are interested in how people solve novel problems. One suggestion is that this is done by mapping knowledge relating to a similar known domain to the new problem-called analogical mapping. Similarities between the known domain and the new one are noted and operators from the known domain are transferred to the new one.

This process has been investigated using analogous stories. Gick and Holyoak gave subjects the following problem:

A doctor is treating a malignant tumor. In order to destroy it he needs to blast it with high-intensity rays. However, these will also destroy the healthy tissue, surrounding tumor. If he lessens the ray's intensity the tumor will remain. How does he destroy the tumor?

The solution to this problem is to fire low-intensity rays from different directions converging on the tumor. That way, the healthy tissue receives harmless low-intensity rays while the tumor receives the rays combined, making a high- intensity does. The investigators found that only 10% of subjects reached this solution without help.

© Copyright Virtual University of Pakistan

However, this rose to 80% when they were given this analogous story and told that it may help them:

A general is attacking a fortress. He can't send all his men in together as the roads are mined to explode if large numbers of men cross them. He therefore splits his men into small groups and sends them in on separate roads. In spite of this, it seems that people often miss analogous information, unless it is semantically close to the problem domain.

Skill acquisition

All of the problem solving that we have considered so far has concentrated on handling unfamiliar problems. However, for much of the time, the problems that we face are not completely new. Instead, we gradually acquire skill in a particular domain area. But how is such skill acquired and what difference does it make to our problem-solving performance? We can gain insight into how skilled behavior works, and how skills are acquired, by considering the difference between novice and expert behavior in given domains.

A commonly studied domain is chess playing. It is particularly suitable since it lends itself easily to representation in terms of problem space theory. The initial state is the opening board position; the goal state is one player checkmating the other; operators to move states are legal moves of chess. It is therefore possible to examine skilled behavior within the context of the problem space theory of problem solving.

In all experiments the behavior of chess masters was compared with less experienced chess players. The first observation was that players did not consider large number of moves in choosing their move, nor did they look ahead more than six moves. Maters considered no mire alternatives than the less experienced, but they took less time to make decision and produced better moves.

It appears that chess masters remember board configurations and good moves associated with them. When given actual board positions to remember, masters are much better at reconstructing the board than the less experienced. However, when given random configurations, the groups of players were equally bad at reconstructing the positions. It seems therefore that expert players 'chunk' the board configuration in order to hold it in short-term memory. Expert player use larger chunks than the less experienced and can therefore remember more detail.

Another observed difference between skilled and less skilled problem solving is in the way that different problems are grouped. Novices tend to group problems according to superficial characteristics such as the objects or features common to both. Experts, on the other hand, demonstrate a deeper understanding of the problems and group them according to underlying conceptual similarities, which may not be at all obvious from the problem descriptions.

© Copyright Virtual University of Pakistan

Lecture 11.

The Psychology of Actions

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand mental models
- Understand psychology of actions
- Discuss errors.

11.1 Mental model

The concept of mental model has manifested itself in psychology theorizing and HCI research in a multitude of ways. It is difficult to provide a definitive description, because different assumption and constraints are brought to bear on the different phenomena it has been used to explain. A well-known definition, in the context of HCI, is provided by Donald Norman: 'the model people have of themselves, others, the environment, and the things with which they interact. People form mental models through experience, training and instruction'.

It should be noted that in fact the term mental model was first developed in the early 1640s by Kenneth Craik. He proposed that thinking '…models, or parallels reality':
'If the organism carries a "small-scale model" of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and future, and in every way to react in a much fuller, safer, and more competent manner to emergencies witch face it.'

Just as an engineer will build scale models of a bridge, in order to test out certain stresses prior to building the real thing, so, too, do we build mental models of the world in order to make predictions about an external event before carrying out an action? Although our construction and use of mental models may not be as extensive or as complete as Craik's hypothesis suggests, it is likely that most of us can probably recall using a form of mental simulation at some time or other. An important observation of these types of mental models is that they are invariably incomplete, unstable, and easily confusable and are often based on superstition rather than scientific fact.

Within cognitive psychology the term mental model has since been explicated by Johnson-Laird (1983, 1988) with respect to its structure and function in human reasoning and language understanding. In terms of structure of mental models, he argues that mental models are either analogical representations or a combination of analogical and prepositional representations. They are distinct from, but related to images. A mental model represents the relative position of a set of objects in an analogical manner that

© Copyright Virtual University of Pakistan

parallels the structure of the state of objects in the world. An image also does this, but more specifically in terms of view of a particular model.

An important difference between images and mental models is in terms of their function. Mental models are usually constructed when we are required to make an inference or a prediction about a particular state of affairs. In constructing the mental model a conscious mental simulation may be 'run' from which conclusions about the predicted state of affairs can be deduced. An image, on the other hand, is considered to be a one-off representation. A simplified analogy is to consider an image to be like a frame in a movie while a mental model is more like a short snippet of a movie.

So, after this discussion we can say that while learning and using a system, people develop knowledge of how to use the system and, to a lesser extent, how the system works. These two kinds of knowledge are often referred to as a user's mental model.

Having developed a mental model of an interactive product, it is assumed that people will use it to make inferences about how to carry out tasks when using the interactive product. Mental models are also used to fathom what to do when something unexpected happens with a system and when encountering unfamiliar systems. The more someone learns about a system and how it functions, the more their mental model develops. For example, TV engineers have a deep mental model of how TVs work that allows them to work out how to fix them. In contrast, an average citizen is likely to have a reasonably good mental model of how to operate a TV but a shallow mental model of how it worked.

To illustrate how we use mental models in our everyday reasoning, imagine the following scenario:

> You arrive home from a holiday on a cold winter's night to a cold house. You have small baby and you need to get the house warm as quickly as possible. Your house is centrally heated. Do you set the thermostat as high as possible or turn it to the desired temperature (e.g., 70F)

Most people when asked the questions imagine the scenario in terms of what they would do in their own house they choose the first option. When asked why, a typical explanation that is given is that setting the temperature to be as high as possible increases the rate at which the room warms up. While many people may believe this, it is incorrect.

There are two commonly held folk theories about thermostats: the timer theory and the valve theory. The timer theory proposes that the thermostat simply controls the relative proportion of time that the device stays on. Set the thermostat midway, and the device is on about half the time; set it all the way up and the device is on all the time; hence, to heat or cool something most quickly, set the thermostat so that the device is on all the time. The valve theory proposes that the thermostat controls how much heat comes out of the device. Turn the thermostat all the way up, and you get maximum heating or cooling.

Thermostats work by switching on the heat and keeping it going at a constant speed until the desired temperature set is reached, at which point they cut out. They cannot control the rate at which heat is given out from a heating system. Left a given setting, thermostats will turn the heat on an off as necessary to maintain the desired temperature. It treats the

© Copyright Virtual University of Pakistan

heater, oven, and air conditioner as all-or-nothing devices that can be either fully on or fully off, with no in-between states. The thermostat turns the heater, oven, or air conditioner completely on—at full power—until the temperature setting on the thermostat is reached. Then it turns the unit completely off. Setting the thermostat at one extreme cannot affect how long it takes to reach the desired temperature.

The real point of the example is not that some people have erroneous theories; it is that everyone forms theories (mental models) to explain what they have observed. In the case of the thermostat the design gives absolutely no hint as to the correct answer. In the absence of external information, people are free to let their imaginations run free as long as the mental models they develop account for the facts as they perceive them.

Why do people use erroneous mental models?

People use erroneous mental models because they rely on familiar but incorrect assumptions about how things work—usually based on past experiences or intuitive beliefs. For example, many people apply the " more is more" logic (turning a tap gives more water, so pressing a button more must work faster), even when it's not appropriate (like with thermostats or elevator buttons). These flawed models are common because:

People generalize from previous experiences.
They tend to seek cause-and-effect relationships, even if none exist.
They often blame themselves when things don't work, due to learned or taught helplessness.
Poor system design reinforces confusion, offering little feedback or clarity.
Many avoid reading manuals or learning how devices actually function.

Improving people's understanding requires either better education or, more effectively, better design—making systems more transparent and intuitive.

It seems that in the above scenario, they are running a mental model based on general valve theory of the way something works. This assumes the underlying principle of "more is more": the more you turn or push something, the more it causes the desired effect. This principle holds for a range of physical devices, such as taps and radio controls, where the more you turn them, the more water or volume is given. However, it does not hold for thermostats, which instead function based on the principle of an on-off switch. What seems to happen is that in everyday life people develop a core set of abstractions about how things work, and apply these to a range of devices, irrespective of whether they are appropriate.

Using incorrect mental models to guide behavior is surprisingly common. Just watch people at a pedestrian crossing or waiting for an elevator (lift). How many times do they press the button? A lot of people will press it at least twice. When asked why, a common reason given is that they think it will make it lights change faster or ensure the elevator arrives. This seems to do another example of following the "more is more" philosophy: it is believed that the more times you press the button; the more likely it is to result in he desire effect.

Another common example of an erroneous mental model is what people do when the cursor freeze on their computer screen. Most people will bash away at all manner of keys in the vain hope that this will make it work again. However, ask them how this will help and their explanations are rather vague. The same is true when the TV starts acting up: a typical response is to hit the top of the box repeatedly with a bare hand or a rolled-up newspaper. Again, as people why and their reasoning about how this behavior will help solve the problem is rather lacking.

Indeed, research has shown that people's mental models of the way interactive devices work is poor, often being incomplete, easily confusable, based on inappropriate analogies, and superstition. Not having appropriate mental models available to guide their behavior is what caused people to become very frustrate—often resulting is stereotypical "venting' behavior like those described above.

On the other hand, if people could develop better mental models of interactive systems, they would be in a better position to know how to carry out their tasks efficiently and what to do if the system started acting up. Ideally, they should be able to develop a mental model that matches the conceptual; modal developed by the designer. But how can you

© Copyright Virtual University of Pakistan

help users to accomplish this? One suggestion is to educate them better, however, many people are resistant to spending much time learning about how things work, especially if it involves reading manuals and other documentation. An alternative proposal is to design systems to be more transparent, so that they are easier to understand.

People do tend to find causes for events, and just what they assign as the cause varies. In part people tend to assign a causal relation whenever two things occur in succession. If I do some action A just prior to some result R, then I conclude that A must have caused R, even if, there really was no relationship between the two.

Self-blaming

Suppose I try to use an everyday thing, but I can't: where is the fault, in my action or in the thing? We are apt to blame ourselves. If we believe that others are able to use the device and if we believe that it is not very complex, then we conclude that any difficulties must be our own fault. Suppose the fault really lies in the device, so that lots of people have the same problems. Because everyone perceives the fault to be his or own, nobody wants to admit to having trouble. This creates a conspiracy of silence, maintaining the feeling of guilt and helplessness among users.

Interestingly enough, the common tendency to blame ourselves for failures with everyday objects goes against the normal attributions people make. In general, it has been found that normal attribute their own problems to the environment, those of other people to their personalities.

It seems natural for people to blame their own misfortunes on the environment. It seems equally natural to blame other people's misfortunes on their personalities. Just the opposite attribution, by the way, is made when things go well. When things go right, people credit their own forceful personalities and intelligence. The onlookers do the reverse. When they see things go well for someone else, they credit the environment.
In all cases, whether a person is inappropriately accepting blame for the inability to work simple objects or attributing behavior to environment or personality, a faulty mental model is at work.

Reason for self-blaming

Learned helplessness

The phenomenon called learned helplessness might help explain the self-blame. It refers to the situation in which people experience failure at a task, often numerous times. As a result, they decide that the task cannot be done, at least not by them: they are helpless. They stop trying. If this feeling covers a group of tasks, the result can be severe difficulties coping with life. In the extreme case, such learned helplessness leads to depression and to a belief that the person cannot cope with everyday life at all. Some times all that it takes to get such a feeling of helplessness is a few experiences that accidentally turn out bad. The phenomenon has been most frequently studied as a precursor to the clinical problem of depression, but it might easily arise with a few bad experiences with everyday life.

© Copyright Virtual University of Pakistan

Taught helplessness

Do the common technology and mathematics phobias results from a kind of learned helplessness? Could a few instances of failure in what appear to be straightforward situations generalize to every technological object, every mathematics problem? Perhaps. In fact, the design of everyday things seems almost guaranteed to cause this. We could call this phenomenon taught helplessness.

With badly designed objects—constructed so as to lead to misunderstanding—faulty mental models, and poor feedback, no wonder people feel guilty when they have trouble using objects, especially when they perceive that nobody else is having the same problems. The problem is that once failure starts, it soon generalizes by self-blame to all technology. The vicious cycle starts: if you fail at something, you think it is your fault. Therefore you think you can't do that task. As a result, next time you have to do the task, you believe you can't so you don't even try. The result is that you can't, just as you thought. You are trapped in a self-fulfilling prophecy.

The nature of human thought and explanation

It isn't always easy to tell just where the blame for problem should be placed. A number of dramatic accidents have come about, in part, from the false assessment of blame in a situation. Highly skilled, well-trained people are using complex equipment when suddenly something goes wrong. They have to figure out what the problem is. Most industrial equipment is pretty reliable. When the instruments indicate that something is wrong, one has to consider the possibility that the instruments themselves are wrong. Often this is the correct assessment. When operators mistakenly blame the instruments for an actual equipment failure, the situation is ripe for a major accident.

It is spectacularly easy to find examples of false assessment in industrial accidents. Analysts come in well after the fact, knowing what actually did happen; with hindsight, it is almost impossible to understand how the people involved could have made the mistake. But from the point of view of the person making decisions at time, the sequence of events is quite natural.

Three Mile Island Nuclear Power Plant

At the Three Mile Island nuclear power plant, operators pushed a button to close a valve; the valve had been opened (properly) to allow excess water to escape from the nuclear core. In fact, the valve was deficient, so it didn't close. But a light on the control panel indicated that the valve position was closed. The light actually didn't monitor the valve, only the electrical signal to the valve, a fact known by the operators. Still, why suspect a problem? The operators did look at the temperature in the pipe leading from the valve: it was high, indicating that fluid was still flowing through the closed valve. Ah, but the operators knew that the valve had been leaky, so the leak would explain the high temperature; but the leak was known to be small, and operators assumed that it wouldn't affect the main operation. They were wrong, and the water that was able to escape from the core added significantly to the problems of that nuclear disaster. Norman says that the operators' assessment was perfectly reasonable: the fault wan is the design of the lights and in the equipment that gave false evidence of a closed valve.

© Copyright Virtual University of Pakistan

Lockheed L-1011

Similarly many airline accidents happened just due to misinterpretations. Consider flight crew of the Lockheed L-1011 flying from Miami, Florida, to Nassau, Bahamas. The plane was over the Atlantic Ocean, about 110 miles from Miami, when the low oil pressure light for one of the three engines went on. The crew turned off the engine and turned around to go back to Miami. Eight minutes later, the low-pressure lights for the remaining two engines also went on, and the instruments showed zero oil pressure and quantity in all three engines. What did the crew do now? They didn't believe it! After all, the pilot correctly said later, the likelihood of simultaneous oil exhaustion in all three engines was "one in millions I would think." At the time, sitting in the airplane, simultaneous failure did seem most unlikely. Even the National Transportation Safety Board declared, "The analysis of the situation by the flight crew was logical, and was what most pilots probably would have done if confronted by the same situation."

What happened? The second and third engines were indeed out of oil, and they failed. So there were no operating engines: one had been turned off when its gauge registered low, the other two had failed. The pilots prepared the plane for an emergency landing on the water. The pilots were too busy to instruct the flight crew properly, so the passengers were not prepared. There was semi-hysteria in the passenger cabin. At the last minute, just as the plane was about to ditch in the ocean, the pilots managed to restart the first engine and land safely to Miami. Then that engine failed at the end of the runway.

Why did all three engine fail? Three missing O-rings, one missing from each of three oil plugs, allowed all the oil to seep out. The O-rings were put in by two different people who worked on the three engines (one for the two plugs on the wings, the other of the plug on the tail). How did both workers make the same mistake? Because the normal method by which they got the oil plugs had been changed that day. The whole tale is very instructive, for there were four major failures of different sorts, from the omission of the O-rings, to the inadequacy of the maintenance procedures, to the false assessment of the problem, to the poor handling of the passengers. Fortunately nobody was injured. The analysts of the National Transportation Safety Board got to write a fascinating report.

Find an explanation, and we are happy. But our explanations are based on analogy with past experience, experience that may not apply in the current situation. In the Three Mile Island incident, past experience with the leaky valve explained away the discrepant temperature reading; on the flight from Miami to Nassau, the pilots' lack of experience with simultaneous oil pressure failure triggered their belief that the instruments must be faulty. Once we have an explanation—correct or incorrect—for otherwise discrepant or puzzling events, there is no more puzzle, no more discrepancy. As a result, we are complacent, at least for a while.

How people do things

To get something done, you have to start with some notion of what is wanted—the goal that is to be achieved. Then, you have to do some thing to the world , that is, take action to move yourself or manipulate someone or something. Finally, you check to see that your goal was made. So there are four different things to consider: the goal, what is done to the world, the world itself, and the check of the world. The action itself has two major aspects: doing something and checking. Call these executions and evaluation Goals do not state precisely what to do—where and how to move, what to pick up. To lead to

© Copyright Virtual University of Pakistan

actions goals must be transformed into specific statements of what is to be done, statements that are called intentions. A goal is some thing to be achieved, often vaguely stated. An intention is specific action taken to get to the goal. Yet even intentions are not specific enough to control actions.

Suppose I am sitting in my armchair, reading a book. It is dust, and the light has gotten dimmer and dimmer. I decide to need more light (that is the goal: get more light). My goal has to be translated into the intention that states the appropriate action in the world: push the switch button on the lamp. There's more: I need to specify how to move my body, how to stretch to reach the light switch, how to extend my finger to push the button (without knocking over the lamp). The goal has to be translated into an intention, which in turn has to make into a specific action sequence, one that can control my muscles. Note that I could satisfy my goal with other action sequences, other intentions. If some one walked into the room and passed by the lamp, I might alter my intention form pushing the switch button to asking the other person to do it for me. The goal hasn't changed, but the intention and resulting action sequence have.

Action Cycle

Human action has two aspects, execution and evaluation. Execution involves doing something. Evaluation is the comparison of what happened in the world with what we wanted to happen

Stages of Execution

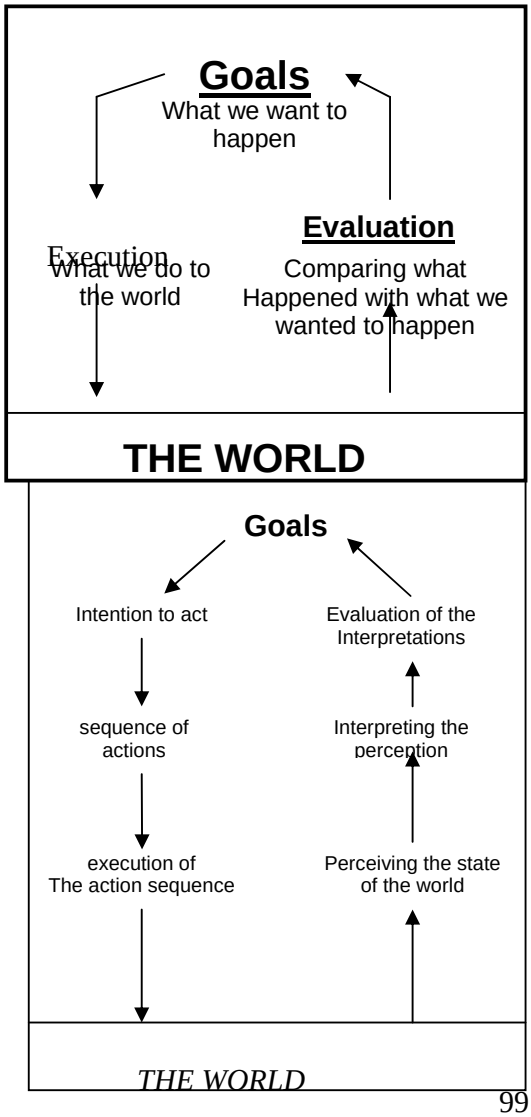
Start at the top with the goal, the state that is to be achieved. The goal is translated into an intention to do some action. The intention must be translated into a set of internal commands, an action sequence that can be performed to satisfy the intention. The action sequence is still a mental event: noting happens until it is executed, performed upon the world.

Stages of Evaluation

Evaluation starts with our perception of the world. This perception must then be interpreted according to our expectations and then compared with respect to both our intentions and our goals

Seven stages of action

The stages of execution (intentions, action sequence, and execution) are coupled with the stages of evaluation (perception, interpretation, and evaluation), with goals common to both stages.



© Copyright Virtual University of Pakistan

11.2 Errors

Human capability for interpreting and manipulating information is quite impressive. However, we do make mistake. Whenever we try to learn a new skill, be it skiing, typing, cooking or playing chess, we are bound to make mistakes. Some are trivial, resulting in no more than temporary inconvenience or annoyance. Other may be more serious, requiring substantial effort to correct. In most situations it is not such a bad thing because the feedback from making errors can help us to learn and understand an activity. When learning to use a computer system, however, learners are often frightened of making errors because, as well as making them feel stupid; they think it can result in catastrophe. Hence, the anticipation of making an error and its consequences can hinder a user's interaction with a system.

Why do we make mistakes and can we avoid them? In order to answer the latter part of the question we must first look at what is going on when we make an error. There are several different types of errors. Some errors result from changes in the context of skilled behavior. If a pattern of behavior has become automatic and we change some aspect of it, the more familiar pattern may break through and cause an error. A familiar example of this is where we intend to stop at the shop on the way home from work but in fact drive past. Here, the activity of driving home is the more familiar and overrides the less familiar intention.

Other errors result from an incorrect understanding, or model, of a situation or system. People build their own theories to understand the casual behavior of systems. These have been termed mental models. They have a number of characteristics. Mental models are often partial: the person does not have a full understanding of the working of the whole system. They are unstable and are subject to change. They can be internally inconsistent, since the person may not have worked through the logical consequences of their beliefs. They are often unscientific and may be based on superstition rather than evidence. Often they are based on an incorrect interpretation of the evidence.

A classification of errors

There are various types of errors. Norman has categorized them into two main types, slips and mistakes:

Mistakes

Mistakes occur through conscious deliberation. An incorrect action is taken based on an incorrect decision. For example, trying to throw the icon of the hard disk into the wastebasket, in the desktop metaphor, as a way of removing all existing files from the disk is a mistake. A menu option to erase the disk is appropriate action.

Slips

Slips are unintentional. They happen by accident, such as making typos by pressing the wrong key or selecting wrong menu item by overshooting. The most frequent errors are slips, especially in well-learned behavior.

© Copyright Virtual University of Pakistan

Lecture 12.

Design Principles

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand conceptual models
- Discuss design principles

Conceptual Model

"The most important thing to design is the user's conceptual model. Every thing else should be subordinated to making that model clear, obvious, and substantial. That is almost exactly the opposite of how most software is designed." (David Liddle)
By a conceptual model is meant:

A description of the proposed system in terms of a set of integrated ideas and concepts about what it should do, behave and look like, that will be understandable by the users in the manner intended.

To develop a conceptual model involves envisioning the proposed product, based on the user's needs and other requirements identified. To ensure that it is designed to be understandable in the manner intended requires doing iterative testing of the product as it is developed.

A key aspect of this design process is initially to decide what the user will be doing when carrying out their tasks. For example, will they be primarily searching for information, creating documents, communicating with other users, recording events, or some other activity? At this stage, the interaction mode that would best supports this need to be considered. For example, would allowing the users to browse be appropriate, or would allowing them to ask questions directly to the system in their native language be more affective? Decision about which kind of interaction style use (e.g., whether to use a menu-based system, speech inputs, commands) should be made in relation to the interaction mode. Thus, decision about which mode of interaction to support differ from those made about which style of interaction to have; the former being at a higher level of abstraction. The former are also concerned with determining the nature of the users' activities to support, while the later are concerned with the selection of specific kinds of interface.

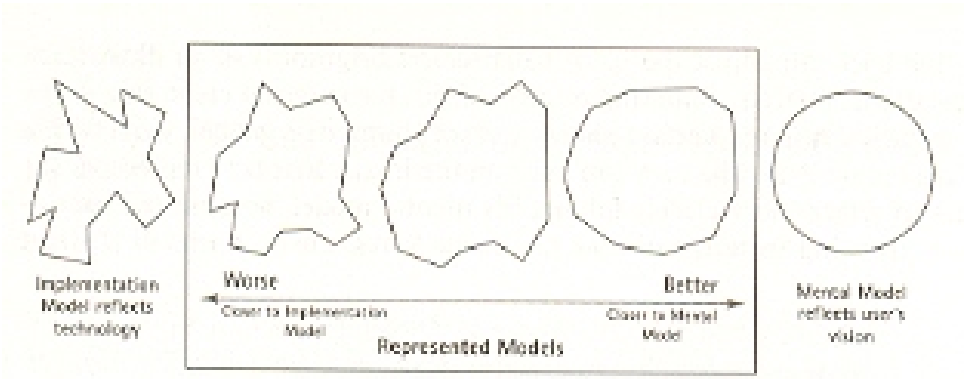
Once a set of possible ways of interacting with interactive system has been identified, the design of the conceptual modal then needs to be thought through in term of actual concrete solution. This entail working out the behavior of the inter face, the particular interaction style that will be used, and the "look and feel" of the interface. At this stage of

© Copyright Virtual University of Pakistan

"fleshing out," it is always a good idea to explore a number of possible designs and to assess the merits and problems of each one.

Another way of designing an appropriate conceptual model is to interface metaphor this can provide a basic structure for the conceptual model that is couched in knowledge users are familiar with. Examples of well-known interface metaphors are the desktop and search engines Software has a behavioral face it shows to the world that is created by the programmer or designer. This representation is not necessarily an accurate description of what is really going on inside the computer, although unfortunately, it frequently is. This ability to represent the computer functioning independent of its true actions is far more pronounced in software than in any other medium. It allows a clever designer to hide some of the more unsavory facts of how the software is really getting the job done. This disconnection between what is implemented and what it offered as explanation gives rise to a third model in the digital world, the designer's represented model—the way the designer chooses to represent a program's functioning to the user. Donald Norman refers to this simply as the designer's model.

In the world of software, a program's represented model can be quite different from the actual processing structure of the program. For example, an operating system can make a network file server look as though it were a local disk. The model does not represent the fact that the physical disk drive may be miles away. This concept of the represented model has no widespread counterpart in the mechanical world. The representation between the three models is shown in Figure.



The closer the represented model comes to the user's mental model, the easier he will find the program to use and to understand. Generally, offering a represented model that follows the implementation model too closely significantly reduces the user's ability to learn and use the program, assuming that the user's mental model of his tasks differs from the implementation model of the software.

We tend to form mental models that are simpler than reality; so if we create represented models that are simpler than the actual implementation model, we help the user achieve a better understanding. Pressing the brake pedal in your car, for example, may conjure a mental image of pushing a lever that rubs against the wheels to slow you down. The actual mechanism includes hydraulic cylinders, tubing, and metal pads that squeeze on a perforated disk, but we simplify all that out of our minds, creating a more effective, albeit less accurate, mental model. In software, we imagine that a spreadsheet scrolls now cells into view when we click on the scrollbar. Nothing of the sort actually happens. There is

© Copyright Virtual University of Pakistan

no sheet of cells out there, but a tightly packed data structure of values, with various pointers between them, from which the program synthesizes a new image to display in real-time.

Another important thing is that there are several gulfs that separate mental states from physical ones. Each gulf reflects one aspect of the distance between the mental representation of the person and the physical components and states of the environment. And these gulfs present major problems for users.

Gulf of Execution is the gap between what a user wants to do and what the system allows.

Gulf of Evaluation is the gap between what the system shows and how easily the user can understand it.

The Gulf of Execution

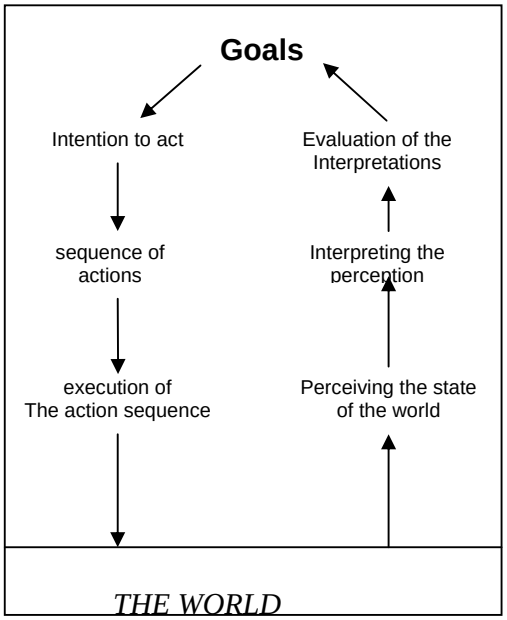
Does the system provide actions that correspond to the intentions of the person? The difference between the intentions and allowable actions is the gulf of execution. One measure of this gulf is how well the system allows the person to do the intended actions directly, without extra effort: do the action provided by the system match those intended by the person?

The Gulf of Evaluation

Does the system provide a physical representation that can be directly perceived and that is directly interpretable in terms of the intentions and expectations of the person? The Gulf of evaluation reflects the amount of effort that the person must exert to interpret the physical state of the system and to determine how well the expectations and intentions have been met. The gulf is small when the system provides information about its state in a form that is easy to get, is easy to interpret, and matches the way the person thinks of the system.

The Seven Stages of Action as Design aids

The seven-stage structure can be a valuable design aid, for it provides a basic checklist of questions to ask to ensure that the Gulfs of evaluation and execution are bridged.
In general each stage of action requires its own special design strategies and, in turn, provides its own opportunity for disaster. It would be fun were it not also so frustrating, to look over the world and gleefully analyze each deficiency. On the whole, as you can see in figure the questions for each stage are relatively simple. And these, in turn, boil down to the principles of good design. Principles of good design are discussed bellow.



12.1 Design Principles

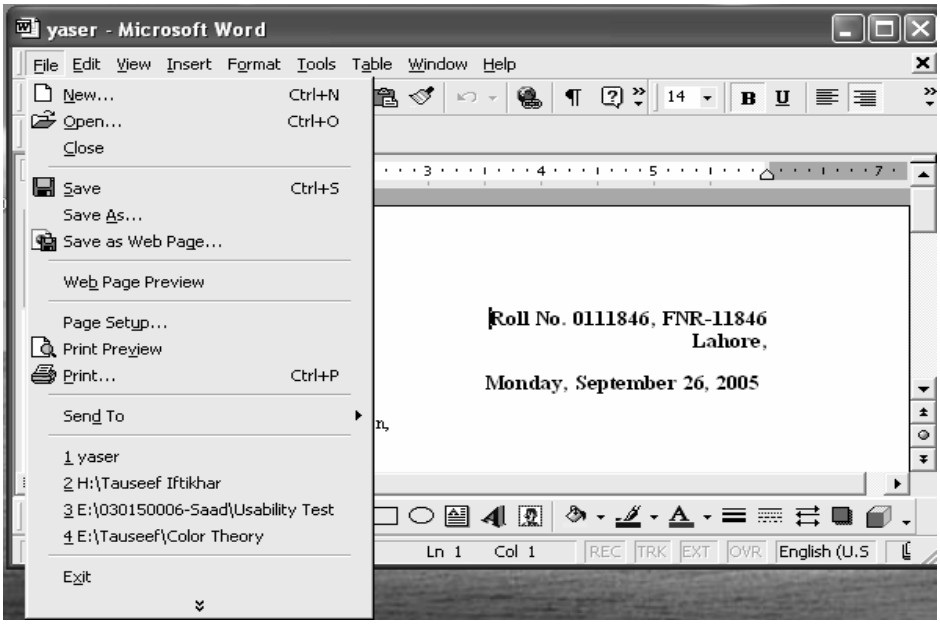
A number of design principles have been promoted. The best known are concerned with how to determine what users should see and do when carrying out their tasks using an interactive product. Here we briefly describe the most common ones

© Copyright Virtual University of Pakistan

- Visibility
- Affordance
- Constraints
- Mapping
- Consistency
- Feedback

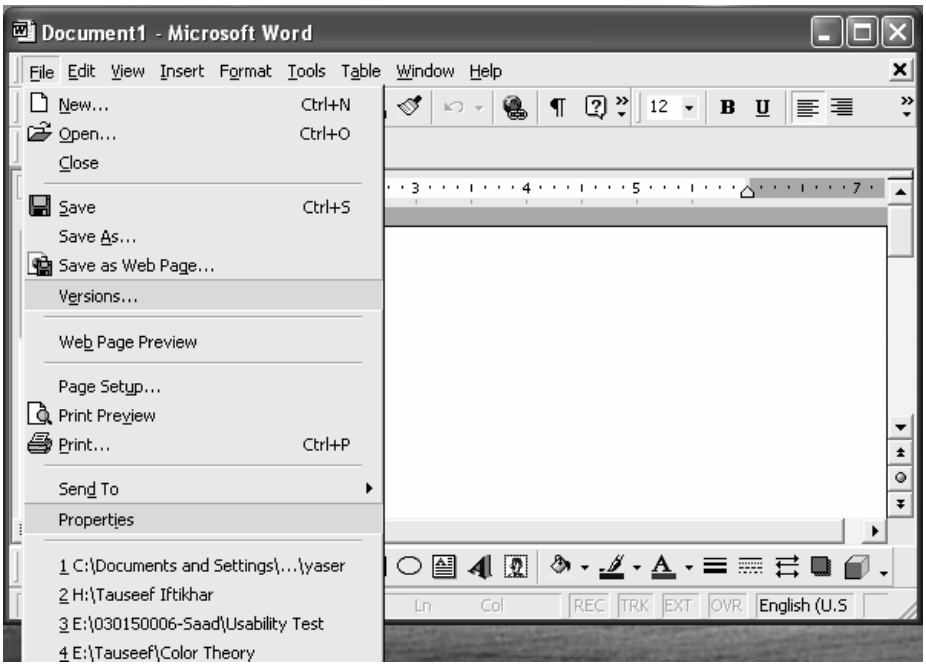
Visibility

The more visible functions are, the more likely users will be able to know what to do next. In contrast, when functions are "out of sight," it makes them more difficult to fid and knows how to use. Norman describes the controls of a car to emphasize this point. The controls for different operations are clearly visible (e.g., indicator, headlights, horn, hazard warning lights), indicating what can be done. The relationship between the way the controls have been positioned in the car and what they do makes it easy for the deriver to find the appropriate control for the task at hand. For example, one problem that I often encounter, in word processing software I often needed to set the properties of a word document. For this logically option of properties should be in the File menu, and I have often seen it there. But once, I opened the file menu I could not find it there, I was confused. Look at the figure



In confusion, I looked through all the menus but in vain. At last, surprisingly I was again looking at the file menu when I noticed the arrow at the bottom of the menu, when I clicked on that I was able to see that option again on the menu. Look at the figure bellow.

© Copyright Virtual University of Pakistan



Affordance

Affordance is a term used to refer to an attribute of an object that allows people to know how to use it. For example, a mouse button invites pushing by the way it is physically constrained in its plastic shell. At a very simple level, to afford means "to give a clue." When the affordances of a physical object are perceptually obvious it is easy to know how to interact with it. For example, a door handle affords pulling, a cup handle affords grasping, and a mouse button affords pushing. Norman introduced this concept in the late 80s in his discussion of the design of everyday objects. Since then, it has been much popularized, being what can be done to them. For example, graphical elements like button, icon, links, and scroll bars are talked about with respect to how to make it appear obvious how they should be used: icons should be designed to afford clicking, scroll bars to afford moving up and down, buttons to afford pushing.

There are two kind of affordance:

- Perceived
- Real

Real

Physical objects are said to have real affordances, like grasping, that are perceptually obvious and do not have to be learned.

Perceived

User interfaces that are screen-based are virtual and do not make sense to try to design for real affordances at the interface---except when designing physical devices, like control consoles, where affordance like pulling and pressing are helpful in guiding the user to

© Copyright Virtual University of Pakistan

know what to do. Alternatively screen based interfaces are better conceptualized as perceived affordances, which are essentially learned conventions.

Constraints are design features that limit user actions to prevent errors and guide correct usage.

Constraints

The design concept of constraining refers to determining ways of restricting the kind of user interaction that can take place at a given moment. There are various ways this can be achieved. A common design practice in graphical user interfaces is to deactivate certain menu options by shading them, thereby restricting the user to only actions permissible at that stage of the activity. One of the advantages of this form of constraining is it prevents the user from selecting incorrect options and thereby refuses the chances of making a mistake. The use of different kinds of graphical representations can also constrain a person's interpretation of a problem or information space. For example flow chart diagram show which objects are related to which thereby constraining the way the information can be perceived. Norman classified constraints into three categories: physical, logical, and cultural.

Physical constraints are limits built into objects based on their shape, size, or design that prevent incorrect use—like a USB only fitting one way.

Physical constraints

Physical constraints refer to the way physical objects restrict the movement of things. For example, the way a external disk can be placed into a disk drive is physically constrained by its shape and size, so that it can be inserted in only one way. Likewise, keys on a pad can usually be pressed in only one way.

Logical constraints use common sense and reasoning to guide users by making it clear what actions make sense or are possible in a given situation.

Logical constraints

Logical constraints rely on people's understanding of the way the world works. They rely on people's common-sense reasoning about actions and their consequences. Picking up a physical marble and placing it in another location on the phone would be expected by most people to trigger something else to happen. Making actions and their effects obvious enables people to logically deduce what further actions are required. Disabling menu options when not appropriate for the task in hand provides logical constraining. It allows users to reason why (or why not) they have been designed this way and what options are available.

Cultural constraints are learned rules or conventions, like red meaning danger, that guide user behavior based on shared cultural understanding.

Culture constraints

Culture constraints rely on learned conventions, like the use of red for warning, the use of certain kinds of signals for danger, and the use of the smiley face to represent happy emotions. Most cultural constraints are arbitrary in the sense that their relationship with what is being represented is abstract, and could have equally evolved to be represented in another form (e.g., the use of yellow instead of red for warning). Accordingly, they have to be learned. Once learned and accepted by a cultural group, they become universally accepted conventions. Two universally accepted interface conventions are the use of windowing for displaying information and the use icons on the desktop to represent operations and documents.

© Copyright Virtual University of Pakistan



Example of logical constraints

Mapping

Mapping is the relationship between a control and its effect, and good mapping makes it easy for users to understand and predict how things work.

This refers to the relationship between controls and their effects in the world. Nearly all artifacts need some kind of mapping between controls and effects, whether it is a flashlight, car, power plant, or cockpit. An example of a good mapping between controls is effect is the up and down arrows used to represent the up and down movement of the cursor, respectively, on a computer keyboard. The mapping of the relative position of controls and their effects is also important. Consider the various musical playing devices. How are the controls of playing rewinding and fast forward mapped onto the desired effects? They usually follow a common convention of providing a sequence of buttons, with the play button in the middle, the rewind button on the left and the fast-forward on the right. This configuration maps directly onto the directionality of the actions.

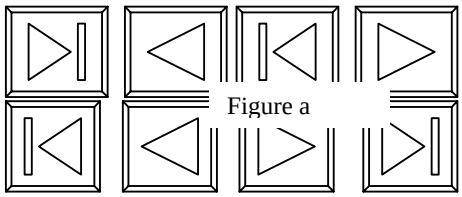


Figure a

Figure b

Imagine how difficult it would be if the mapping in figure (a) were used.

Consistency

Consistency means designing interfaces where similar tasks are done in the same way, helping users learn and use them more easily—like always using the left click to select items.

This refers to designing interfaces to have similar operations and use similar elements for achieving similar tasks. In particular, a consistent interface is one that follows rules, such as using the same operation to select all objects. For example, a consistent operation is using the same input action to highlight any graphical object at the interfaces, such as always clicking the left mouse button. Inconsistent interfaces, on the other hand, allow

© Copyright Virtual University of Pakistan

exceptions to a rule. An example of this is where certain graphical objects (e.g., email messages presented in a table) can be highlighted using the right mouse button, while all other operations are highlighted using the left button. A problem with this kind of inconsistency is that is quite arbitrary, making it difficult for users to remember and making the users more prone to mistakes.

On of the benefits of consistent interfaces, therefore, is that they are easier to learn and use. Users have to learn only a single mode of operation that is applicable to all objects. This principle worked well for simple interfaces with limited operations, like mini CD player with small number of operations mapped onto separate buttons. Here all the user has to do is learn what each button represents and select accordingly. However, it can be more problematic to apply the concept of consistency to more complex interfaces, especially when many different operations need to be designed for. For example, consider how to design an interface for an application that offers hundreds of operations. There is simply not enough space for a thousand buttons, each of which maps onto an individual operation. Even if there were, it would be extremely difficult and time consuming for the user to search through them all to find the desired operation.

A much more effective design solution is to create categories of commands that can be mapped into subsets of operations. For the word-processing application, the hundreds of operation available are categorized into subsets of different menus. All commands that are concerned with file operations are placed together in the same file menu.

Another problem with consistency is determining what aspect of an interface to make consistent with what else. There are often many choices, some of which can be inconsistent with other aspects of the interface or ways of carrying out actions. Consider the design problem of developing a mechanism to let users lock their files on a shared server. Should the designer try to design it to be consistent with the way people lock things in the outside world (called external consistency) or with the way they lock objects in the existing system (called internal consistency)? However, there are many different ways of locking objects in the physical world (e.g., placing in a safe, using a padlock, using a key, using a child safety lock), just as there are different ways of locking electronically. The problem facing designer knows which one to be consistent with.

Feedback

Feedback is the system ' s way of showing the user what action has been performed and what effect it had, helping them continue smoothly.

Related to the concept of visibility is feedback. This is best illustrated by an analogy to what everyday life would be like without it. Imagine trying to play a guitar, slice bread using knife, or write a pen if none of the actions produced any effect for several seconds. There would be an unbearable delay before the music was produced, the bread was cut, or the words appeared on the paper, making it almost impossible for the person to continue with the next strum, saw, or stroke.

Feedback is about sending back information about what action has been done and what has been accomplished, allowing the person to continue with the activity. Various kinds of feedback are available for interaction design—audio, tactile, verbal, visual, and combinations of these. Deciding which combinations are appropriate for different kinds of activities and interactivities is central. Using feedback in the right way can also provide the necessary visibility for user interaction.

© Copyright Virtual University of Pakistan

Lecture 13.

The Computer

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

Describe the advantages and disadvantages of different input output devices keeping in view different aspects of HCI

In previous lectures our topics of discussion were covering the human aspects. From now we will pay some attention towards computers. We will study some computer aspects. You may have studied many of them before in any other course, but that are also part of our discussion, as at one side of our subject is human and at the other side computer lies.
Today will look at some input and output devices of computer. Let us fist look at input devices.

13.1 Input devices

Input is concerned with recording and entering data into computer system and issuing instruction to the computer. In order to interact with computer systems effectively, users must be able to communicate their interaction in such a way that the machine can interpret them. Therefore, input devices can be defined as: a device that, together with appropriate software, transforms information from the user into data that a computer application can process.
One of the key aims in selecting an input device and deciding how it will be used to control events in the system is to help users to carry out their work safely, effectively, efficiently and, if possible, to also make it enjoyable. The choice of input device should contribute as positively as possible to the usability of the system. In general, the most appropriate input device will be the one that:

- Matches the physiology and psychological characteristics of users, their training and their expertise. For example, older adults may be hampered by conditions such as arthritis and may be unable to type; inexperienced users may be unfamiliar with keyboard layout.
- Is appropriate for the tasks that are to be performed. For example, a drawing task from a list requires an input device that allows continuous movement; selecting an option from a list requires an input device that permits discrete movement.
- Is suitable for the intended work and environment. For example, speech input is useful where there is no surface on which to put a keyboard but is unsuitable in noisy condition; automatic scanning is suitable if there is a large amount of data to be generated.

Frequently the demands of the input device are conflicting, and no single optimal device can be identified: trade-offs usually have to be made between desirable and undesirable features in any given situation. Furthermore, many systems will use two or more input

© Copyright Virtual University of Pakistan

devices together, such as a keyboard and a mouse, so the devices must be complementary and well coordinated. This means that not only must an input device be easy to use and the form of input be straightforward, there must also be adequate and appropriate system feedback to guide, reassure, inform and if necessary, correct user's errors. This feedback can take various forms. It can be a visual display screen: a piece of text appears, an icon expands into a window, a cursor moves across the screen or a complete change of screen presentation occurs. It can be auditory: an alarm warning, a spoken comment or some other audible clue such as the sound of keys clicking when hit. It can be tactile: using a joystick. In many cases feedback from input can be a combination of visual, auditory and tactile responses. For example, when selecting an icon on a screen, the tactile feedback from the mouse button or function keys will tell users that they instructed the system to activate the icon. Simultaneously, visual feedback will show the icon changing shape on the screen. This is coordinated with the sound of the button clicking or the feel of the key resisting further pressure. Let us now discuss various types of devices in terms of their common characteristics and the factors that need to be considered when selecting an input device. We will discuss text entry devices first.

13.2 Text entry devices

There are many text entry devices as given below:

Keyboard

The most common method of entering information into the computer is through a keyboard. Since you have probably used them a lot without perhaps thinking about the related design issue, thinking about keyboards is a convenient starting point for considering input design issue. Broadly defined, a keyboard is a group of on—off push button, which are used either in combination or separately. Such a device is a discrete entry device. These devices involve sensing essentially one of two or more discrete positions (for example, keys on keyboards, touch-sensitive switches and buttons), which are either on or off, whereas others (for example, pens with digitizing tablets, moving joysticks, roller balls and sliders) involve sensing in a continuous range. Devices in this second category are therefore, known as continuous entry devices.

When considering the design of keyboards, both individual keys and grouping arrangements need to be considered. The physical design of keys is obviously important. For example, of keys are too small this may cause difficulty in locating and hitting chosen keys accurately. Some calculators seeking extreme miniaturization and some modern telephones suffer from this. Some keyboards use electro mechanical switches, while others use sealed, flat membrane keyboards. When pressing a key on a membrane keyboard, unless appropriate feedback is given on screen, or using sound it may be difficult to tell which key , if any , has been presses. On the other hand, membrane keyboards can typically withstand grease, dirt and liquids that would soon clog up typical electromechanical switches. This can be an important consideration in environments such as production floors, farm and public places.

Alterations in the arrangement of the keys can affect a user's speed and accuracy. Various studies have shown that typing involves a great deal of analyses of trained typists suggest that typing is not a sequential act, with each key being sought out and pressed as the letters occur in the works to be typed. Rather, the typist looks ahead, processes text in chunks, and then types it in chunks. For alphabetic text these chunks are about two to

© Copyright Virtual University of Pakistan

three world long for numerical material they are three to four characters long. The effect is to increase the typing speed significantly.

QWERTY keyboard

Most people are quite familiar with the layout of the standard alphanumeric keyboard, often called the qwerty keyboard, the name being derived from the first letters in the upper most row from left to center. This design first became a commercial success when used for typewriters in the USA in 1874, after many different prototypes had been tested. The arrangement of keys was chosen in order to reduce the incidence of keys jamming in the manual typewriters of the time rather than because of any optimal arrangement for typing. For example, the letters 's', ,t, and 'h' are far apart even though they are far apart even though they are frequently used together.

Alphabetical Keyboard:

Keys are arranged A to Z in order.

Not commonly used because it ' s not efficient for typing— you have to move fingers more.

Easy to learn but very slow for typing.

Dvorak Keyboard:

Designed to place the most used letters under the strongest fingers.

Common letters like A, O, E, U, I, D, H, T, N , and S are on the home row.

It reduces finger movement and increases typing speed and accuracy.

Alphabetic keyboard

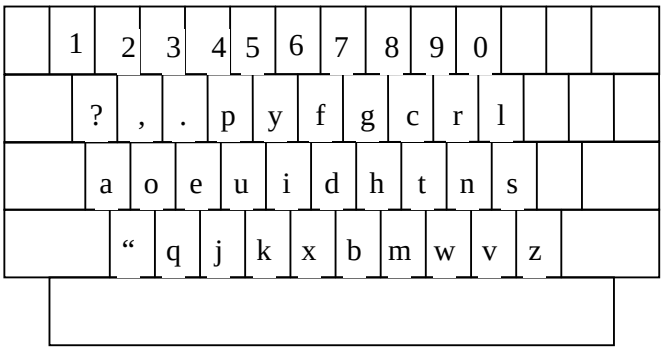
One of the most obvious layouts to be produced is the alphabetic keyboard, in which the letters are arranged alphabetically across the keyboard. It might be expected that such a layout would make it quicker for untrained typists to use, but this is not the case. Studies have shown that this keyboard is not faster for properly trained typists, as we may expect, since there is no inherent advantage to this layout. And even for novice or occasional users, the alphabetic layout appears to make very little difference to the speed of typing. These keyboards are used in some pocket electronic personal organizers, perhaps because the layout looks simpler to use than the QWERTY one. Also, it dissuades people from attempting to use their touch-typing skills on a very small keyboard and hence avoids criticisms of difficulty of use.

Dvorak Keyboard

With the advent of electric and electronic keyboards and the elimination of levered hammers such considerations are no longer necessary. Attempts at designing alternative keyboards that are more efficient and quicker to use have produced, among others, the Dvorak and Alphabetic boards. The Dvorak board, first patented in 1932, was designed using the following principles:

- Layout is arranged on the basis of frequency of usage of letters and the frequency of letter pattern and sequences in the English language.
- All vowels and the most frequently used consonants are on the second or home row, so that something likes 70% of common words are typed on this row alone.
- Faster operation is made possible by tapping with fingers on alternate hands (particularly the index fingers) rather than by repetitive tapping with one finger and having the majority of keying assigned to one hand, as in the QWERTY keyboard, which favors left-handers. Since the probability of vowels and consonants altering is very high, all vowels are typed with the left hand and frequent home row consonants with the right.

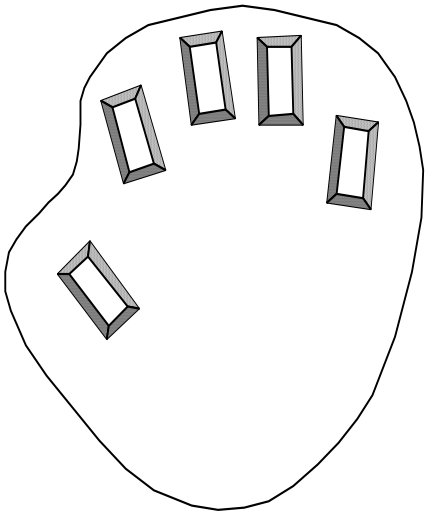
© Copyright Virtual University of Pakistan



The improvements made by such as ergonomic design are a significant reduction in finger travel and consequent fatigue and a probable increase in accuracy. Dvorak also claimed that this arrangement reduces the between –row movement by 90% and allows 35% of all words normally used to be typed on the home row. Despite its significant benefits, the dvorak layout, show in figure has never been commercially successful. The possible gain in input speed has to be weighed against the cost of replacing existing keyboards and retraining millions of people who have learned the QWERTY keyboard.

Chord keyboards

In chord keyboards several keys must be pressed at once in order to enter a single character. This is a bit like playing a flute, where several keys must be pressed to produced with a small number of keys, few keys are required, so chord keyboards can be very small, and many can be operated with just one hand. Training is required learn the finger combination required to use a chord keyboard. They can be very useful where space is very limited, or where one hand is involved in some other task. Training is required to learn the finger combinations required to use a chord keyboard. They can be very useful where space is very limited, or where one hand is involved in some other task. Chord keyboards are also used for mail sorting and a form of keyboard is used for recording transcripts of proceeding in law courts.



Some keyboards are even made of touch-sensitive buttons, which require a light touch and practically no travel; they often appear as a sheet of plastic with the buttons printed on them. Such keyboards are often found on shop till, though the keys are not QWERTY, but specific to the task. Being fully sealed, they have the advantage of being easily cleaned and resistant to dirty environment, but have little feel, and are not popular with trained touch-typists. Feedback is important even at this level of human-computer interaction! With the recent increase of repetitive strain injury (RSI) to users’ finger, and the increased responsibilities of employers in these circumstances, it may be that such

© Copyright Virtual University of Pakistan

designs will enjoy resurgence in the near future. The tendons that control the movement of the fingers becoming inflamed owing to overuse cause RSI in fingers and making repeated unnatural movement.

There are very verities of specially shaped keyboards to relieve the strain of typing or to allow people to type with some injury or disability. These may slope the keys towards the hands to improve the ergonomics position, be designed for single-handed use, or for no hands at all. Some use bespoke key layouts to reduce strain of finger movements. The keyboard illustrated is produced by PCD Maltron Ltd. for left-handed use.

Phone pad and T9 entry

With mobile phones being used for SMS text messaging and WAP, the phone keypad has become an important form of text input. Unfortunately a phone only has digits 0-9, not a full alphanumeric keyboard.

To overcome this for text input the numeric keys are usually pressed several times. Figure shows a typical mapping of digits to letters. For example, the 3 keys have ‘def’ on it. If you press the key once you get a ‘d’, if you press 3 twice you get an ‘e’, and if you press it three times you get an ‘f’. The main number-to-letter mapping is standard, but punctuation and accented letters differ between phones. Also there needs to be a way for the phone to distinguish, say, the ‘dd’ from ‘e’. on some phones you need to pause far short period between successive letters using the same key, for others you press an additional key (e.g. ‘#’).

Most phones have at least two modes for the numeric buttons: one where the keys mean the digits (for example when entering a phone number) and one where they mean letters (for example when typing an SMS message). Some have additional modes to make entering accented characters easier. Also a special mode or setting is needed for capital letters although many phones use rules to reduce this, for example automatically capitalizing the initial letter in a message and letters following full stops, question marks and exclamation marks.

This is all very laborious but you can see experienced mobile users make use of highly developed shorthand to reduce the number of keystrokes. If you watch a teenager or other experienced txt-er, you will see they often develop great typing speed holding the phone in one hand and using only their thumb. As these skills spread through society it may be that future devices use this as a means of small format text input. For those who never develop this physical dexterity some phones have tiny plug-in keyboards, or come with foldout keyboards.

Another technical solution to the problem is the T9 algorithm. This uses a large dictionary to disambiguate words by simply typing the relevant letters once. For example, ‘3926753’ becomes ‘example’ as there is only one word with letters that match (alternative like ‘ewbosld’ that also match is not real words). Where there are ambiguities such as ‘26’, which could be an ‘am’ or an ‘an’, the phone gives a series of option to choose from.

Handwriting recognition

Handwriting is a common and familiar activity, and is therefore attractive as a method of text entry. If we were able to write as we would when we use paper, but with the

© Copyright Virtual University of Pakistan

computer taking this form of input and converting it to text, we can see that it is an intuitive and simple way of interacting with the computer. However, there are a number of disadvantages with hand writing recognition. Current technology is still fairly inaccurate and so makes a significant number of mistakes in recognizing letters, though it has improved rapidly. Moreover, individual differences in handwriting are enormous, and make te recognition process even more difficult. The most significant information in handwriting is not in the letter shape itself but in the stroke information– the way in which the letter is drawn. This means that devices which support handwriting recognition must capture the stoke information, not just the final character shape. Because of this, online recognitions far easier than reading handwritten text on paper. Further complications arise because letters within words are shaped and often drawn very differently depending on the actual vide enough information. More serious in many ways is the limitation on speed; it is difficult to write at more than 25 words a minute, which is no more than half the speed of a decent typist.

The different nature of handwriting means that we may find it more useful in situation where a keyboard-based approach would have its own problems. Such situation will invariably result in completely new systems being designed around the handwriting recognizer as the predominant mode of textural input, and these may bear very little resemblance to the typical system. Pen-based systems that use handwriting recognition are actively marked in the mobile computing market, especially for smaller pocket organizers. Such machines are typically used for taking notes and jotting down and sketching ideas, as well as acting as a diary, address book and organizer. Using handwriting recognition has many advantages over using a keyboard. A pen-based system can be small and yet still accurate and easy to use, whereas small keys become very tiring, or even impossible, to use accurately. Also the pen-based approach does not have to be altered when we move from jotting down text to sketching diagrams; pen-based input is highly appropriate for this also.

Some organizer designs have dispensed with a keyboard completely. With such systems one must consider all sorts of other ways to interact with the system that are not character based. For example, we may decide to use gesture recognition, rather than commands, to tell the system what to do, for example, drawing a line through a word in order to delete it. The important point is that a different input device that was initially considered simply as an alternative to the keyboard opens up a whole host of alternative designs and different possibilities for interaction.

Speech recognition

Speech recognition is a promising are of text entry, but it has been promising for a number of years and is still only used in very limited situations. However, speech input suggests a number of advantages over other input methods:

- Since speech is a natural form of communication, training new users is much easier than with other input devices.
- Since speech input does not require the use of hands or other limbs, it enables operators to carry out other actions and to move around more freely.
- Speech input offers disabled people such as the blind and those with severs motor impairment the opportunities to use new technology.

© Copyright Virtual University of Pakistan

However, speech input suffers from a number of problems:

- Speech input has been applied only in very specialized and highly constrained tasks.
- Speech recognizers have severe limitations whereas a human would have a little problem distinguishing between similar sounding words or phrases; speech recognition systems are likely to make mistakes.
- Speech recognizers are also subject to interference from background noise, although the use of a telephone-style handset or a headset may overcome this.
- Even if the speech can be recognized, the natural form of language used by people is very difficult for a computer to interpret.

The development of speech input systems can be regarded as a continuum, with device that have a limited vocabulary and recognize only single words at one end of the spectrum and systems that attempt to understand natural speech at the other, Isolated word recognition systems typically require pauses between words to be longer than in natural speech and they also tend to be quite careful about how she speaks. Continuous speech recognition systems are capable, up to a point, of problems and system complexity. Although these systems still operate by recognizing a restricted number of words, the advantage is that they allow much faster data entry and are more natural to use. One way of reducing the possible confusion between words is to reduce the number of people who use the system. This can overcome some of the problem caused by variations in accent and intonation. Speaker-dependent systems require each user to train a system to recognize her voice by repeating all the words in the desired vocabulary one or more times. However, individual variability in voice can be a problem, particularly when a user has a cold. It is not uncommon for such systems to confuse words like three and repeat. Speaker-independent systems, as the name suggests, do not have this training requirement; they attempt to accommodate a large range of speaking characteristics and vocabulary. However, the problem of individual variability means that these types of system are less reliable, or have a smaller vocabulary than speaker-dependent systems.

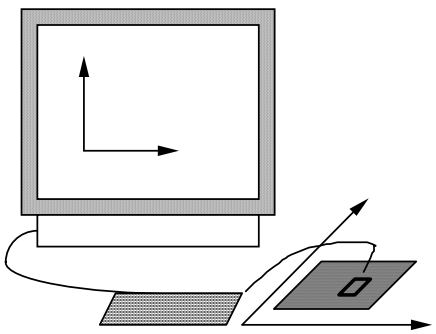
The perfect system would be one that would understand natural speech to such extent that it could not only distinguish differences in speech presentation but also have the intelligence to resolve any conflicts in meaning by interpreting speech in relation to the context of the conversation, as a human being does. This is a deep unsolved problem in Artificial Intelligence, and progress is likely to be slow.

13.3 Positioning, Pointing and Drawing

Pointing devices are input devices that can be used to specify a point or path in a one-, two- or three- dimensional space and, like keyboards, their characteristics have to be considering in relation to design needs.

Pointing devices are as follow:

- Mouse
- Touch pad
- Track ball
- Joystick
- Touch screen
- Eye gaze



© Copyright Virtual University of Pakistan

Mouse

The mouse has become a major component of the majority of desktop computer systems sold today, and is the little box with the tail connecting it to the machine in our basic computer system picture. It is a small, palm-sized box housing a weighted ball- as the box is moved on the tabletop; the ball is rolled by the table and so rotates inside the housing. This rotation is detected by small rollers that are in contact with the ball, and these adjust the values of potentiometers.

The mouse operates in a planar fashion, moving around the desktop, and is an indirect input device, since a transformation is required to map from the horizontal nature of desktop to the vertical alignment of the screen. Left-right motion is directly mapped, whilst up-down on the screen is achieved by moving the mouse away-towards the user.

Foot mouse

Although most mice are hand operated, not all are there have been experiments with a device called the foot mouse. As the name implies, it is foot-operated device, although more akin to an isometric joysticks than a mouse. The cursor is moved by foot pressure on one side or the other of pad. This allows one to dedicate hands to the keyboard. A rare device, the foot mouse has not found common acceptance.

Touch pad



Touchpad's are touch-sensitive tablets usually around 2-3 inches square. They were first used extensively in Apple PowerBooks portable computers but are now used in many other notebook computers and can be obtained separately to replace the mouse on the desktop. They are operated by stroking a finger over their surface, rather like using a simulated trackball. The feel is very different from other input devices, but as with all devices users quickly get used to the action and become proficient.

Because they are small it may require several strokes to move the cursor across the screen. This can be improved by using acceleration settings in the software linking the trackpad movement to the screen movement. Rather than having a fixed ratio of pad distance to screen distance, this varies with the speed of movement. If the finger moves slowly over the pad then the pad movements map to small distances on the screen. If the finger is moving quickly the same distance on the touchpad moves the cursor a long distance.

Trackball and thumbwheel

Trackball is really just an upside-down mouse. A weighted ball faces upwards and is rotated inside a static housing, the motion being detected in the same way as for a mechanical mouse, and the relative motion of the ball moves the cursor. It is a very compact device, as it requires no additional space in which to operate. It is an indirect device, and requires separate buttons for selection. It is fairly accurate, but is hard to draw with, as long movements are difficult. Trackball now appear in a wide variety of sizes, the most usual being about the same as golf ball, with a number of larger and smaller devices available.

© Copyright Virtual University of Pakistan

Thumbwheels are different in that they have two orthogonal dials to control the cursor position. Such a device is very cheap, but slow, and it is difficult to manipulate the cursor in any way other than horizontally or vertically. This limitation can sometimes be a useful constraint in the right application.

Although two-axis thumbwheels are not heavily used in mainstream applications, single thumbwheels are often included on a standard mouse in order to offer an alternative mean to scroll documents. Normally scrolling requires you to grab the scroll bar with the mouse cursor and drag it down. For large documents it is hard to be accurate and in addition the mouse dragging is done holding a finger down which adds to hand strain. In contrast the small scroll wheel allows comparatively intuitive and fast scrolling, simply rotating the wheel to move the page.

Joystick and track point

The joystick is an indirect input device, taking up very little space. Consisting of a small palm-sized box with a stick or shaped grip sticking up form it, the joystick is a simple device with which movements of the stick cause a corresponding movement of the screen cursor. There are two type of joystick: the absolute and the isometric.

In absolute joystick, movement is the important characteristic, since the position of the joystick in the base corresponds to the position of the cursor on the screen. In the isometric joystick, the pressure on the stick corresponds to the velocity of the cursor, and when released, the stick returns to its usual upright centered position.

Track point is a smaller device but with the same basic characteristics is used on many laptop computers to control the cursor. Some older systems had a variant of this called the key mouse, which was a single key. More commonly a small rubber nipple projects in the center of keyboard and acts as a tiny isometric joystick. It is usually difficult for novice to use, but this seems to be related to fine adjustment of the speed settings.

Touch screens

Touch displays allow the user to input information into the computer simply by touching an appropriate part of the screen or a touch-sensitive pad near to the screen. In this way the screen of the computer becomes a bi-directional instrument in that it both receives information from a user and displays output from a system. Using appropriate software different parts of a screen can represent different responses as different displays are presented to a user. For example, a system giving directions to visitors at a large exhibition may first present an overview of the exhibition layout in the form of general map. A user may then be requested to touch the hall that he wishes to visit and the system will present a list of exhibits. Having selected the exhibit of his choice by touching it, the user may then be presented with a more detailed map of the chosen hall.

The advantages of touch screens are that they are easy to learn, require no extra workplace, have no moving parts and are durable. They can provide a very direct interaction. Ease of learning makes them ideal for domains in which use by a particular user may occur only once or twice, and users cannot be expected to spend a time learning to use the system.

© Copyright Virtual University of Pakistan

They suffer from a number of disadvantages, however. Using the finger to point is not always suitable, as it can leave greasy marks on screens and, being a fairly blunt instrument, it is quite inaccurate. This means that the selection of small regions is very difficult, as is accurate drawing. Moreover, lifting the arm to point a vertical screen is very tiring, and also means that the screen has to be within about a meter of the user to enable to be reached, which can make it too close for comfort.

Stylus and light pen

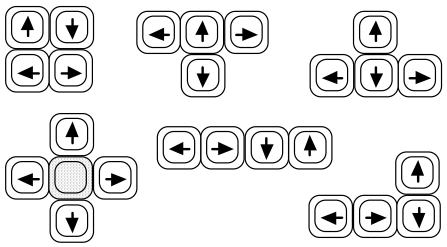
For more accurate positioning, systems with touch-sensitive surface often employ a stylus. Instead of pointing at the screen directly, small pen-like plastic stick is used to point and draw on the screen. This is particularly popular in PDAs, but they are also being used in some laptop computers.

An old technology that is used in the same way is the light pen. The pen is connected to the screen by a cable and, in operation, is held to the screen and detects a burst of light from the screen phosphor during the display scan. The light pen can therefore address individual pixels and so is much more accurate than the touch screen.

Eyegaze

Eyegaze systems allow you to control the computer by simply looking at it. Some systems require you to wear special glasses or a small head-mounted box, others are built into the screen or sit as a small box below the screen. A low-power laser is shone into the eye and is reflected off the retinal. The reflection changes as the angle of the eye alters, and by tracking the reflected beam the eyegaze system can determine the direction in which the eye is looking. The system needs to be calibrated, typically by staring at a series of dots on the screen, but thereafter can be used to move the screen cursor or for other more specialized uses. Eyegaze is a very fast and accurate device, but the more accurate versions can be expensive. It is fine for selection but not for drawing since the eye does not move in smooth lines. Also in real application it can be difficult to distinguish deliberately gazing at some thing and accidentally glancing it.

Cursor keys



Cursor keys are available on most keyboards. Four keys on the keyboard are used to control the cursor, one each for up, down, left and right. There is no standardized layout for the keys. Some layouts are shown in figure but the most common now is the inverted ‘T’.
Cursor keys used to be more heavily used in character-based systems before windows and mice were the norm. However, when logging into remote machines such as web servers, the interface is often a virtual character-based terminal within a telnet window.

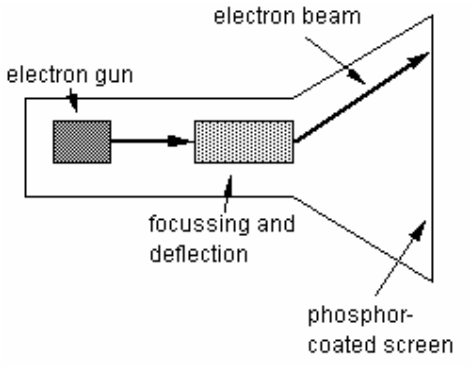
13.4 Display devices

Cathode ray tube

The cathode ray tube is the television-like computer screen still most common as we write this, but rapidly being displaced by flat LCD screens. It works in a similar way to a

© Copyright Virtual University of Pakistan

standard television screen. A stream of electrons is emitted from an electron gun, which is then focused and directed by magnetic fields. As the beam hits the phosphor-coated screen, the phosphor is excited by the electrons and glows. The electron beam is scanned from left to right, and then flicked back to rescan the next line, from top to bottom.



Black and white screens are able to display grayscale by varying the intensity of the electron beam; color is achieved using more complex means. Three electron guns are used, one each to hit red, green and blue phosphors. Combining these colors can produce many others, including white, when they are all fully on. These three phosphor dots are focused to make a single point using a shadow mask, which is imprecise and gives color screens a lower resolution than equivalent monochrome screens.

The CRT is a cheap display device and has fast enough response times for rapid animation coupled with a high color capability. Note that animation does not necessarily means little creatures and figures running about on the screen, but refers in a more general sense to the use of motion in displays: moving the cursor, opening windows, indicating processor-intensive calculations, or whatever. As screen resolution increased, however, the price rises. Because of the electron gun and focusing components behind the screen, CRTs are fairly bulky, though recent innovations have led to flatter displays in which the electron gun is not placed so that it fires directly at the screen, but fires parallel to the screen plane with the resulting beam bent through 90 degrees to his the screen.

Liquid Crystal Display

Liquid Crystal Displays are mostly used in personal organizer or laptop computers. It is a light, flat plastic screen. These displays utilize liquid crystal technology and are smaller, lighter and consume far less power than traditional CRTs. These are also commonly referred to as flat-panel displays. They have no radiation problems associated with them, and are matrix addressable, which means that individual pixels can be accessed without the need for scanning.

This different technology can be used to replace the standard screen on a desktop computer, and this is now common. However, the particular characteristics of compactness, lightweight, and low power consumption have meant that these screens have created a large niche in the computer market by monopolizing the notebook and portable computer systems side.

© Copyright Virtual University of Pakistan

Digital paper

A new form of display that is still in its infancy is the various forms of digital papers. These are thin flexible materials that can be written to electronically, just like a computer screen, but which keep their contents even when removed from any electrical supply.

Physical controls and sensors

Sound output refers to using auditory signals, like beeps or clicks, to provide feedback and convey information, enhancing user interaction with the system.

Sound output

Another mode of output that we should consider is that of auditory signals. Often designed to be used in conjunction with screen displays, auditory outputs are poorly understood: we do not yet know how to utilize sound in a sensible way to achieve maximum effect and information transference. Sounds like beeps, bongs, clanks, whistles and whirrs are all used for varying effect. As well as conveying system output, sounds offer an important level of feedback in interactive systems. Keyboards can be set to emit a click each time a key is pressed, and this appears to speed up interactive performance. Telephone keypads often sound different tones when the keys are pressed; a noise occurring signifies that the key has been successfully pressed, whilst the actual tone provides some information about the particular key that was pressed.

13.5 Touch, feel and smell

Sense of touch and feel is also used for feedback; tactile feedback has its own importance and is being used in many interactive devices. We usually feel textures when we move our fingers over a surface. Technology for this is just beginning to become available.

13.6 Physical controls

A desktop computer has to serve many functions and do has generic keys and controls that can be used for a variety of purpose. In contrast, these dedicated controls panes have been designed for a particular device and for a single use. This is why they differ so much.

Usually microwave a flat plastic control panel. The reason is this, the microwave is used in the kitchen whilst cooking, with hands that may be greasy or have food on them. The smooth controls have no gaps where food can accumulate and clog buttons, so it can easily be kept clean an hygienic.

When using the washing machine you are handling dirty clothes, which may be grubby, but not to the same extent, so the smooth easy-clean panel is less important. It has several major settings and the large buttons act both as control and display.

13.7 Environment and bio sensing

Although we are not always conscious of them, there are many sensors in our environment—controlling automatic doors, energy saving lights, etc. and devices monitoring our behavior such as security tags in shops. The vision of ubiquitous computing suggests that our world will be filled with such devices.

© Copyright Virtual University of Pakistan

Lecture 14.

Interaction

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Define interaction
- Discuss interaction styles keeping in view different aspects of HCI

In the previous lectures we have studied the detailed introduction of human side and computer side. These are two participant of our course Human Computer Interaction. As the name of the course reveals that both of these complex entities are not in isolation rather they come in contact with each other. Human communicate with computers.
There are a number of ways in which human can communicate with the system. If we look at the beginning, batch input system was used, in which the user provides all the information to the computer in form of batch. Now a day it is the age of virtual reality and ubiquitous computing. Here user constantly interacts with computers in his surroundings. Today there is richer interaction.

14.1 The terms of Interaction

Domain

A domain defines an area of expertise and knowledge in some real-world activity. Some examples of domains are graphic design, authoring and process control in a factory. A domain consists of concepts that highlight its important aspects. In a graphic design domain, some of the important concepts are geometric shapes, a drawing surface and a drawing utensil.

Task

Tasks are the operation to manipulate the concepts of a domain. A goal is the desired output from a performed task. For example, one task within the graphic design domain is the construction of a specific geometric shape with particular attributes on the drawing surface.

Goal

A related goal would be to produce a solid red triangle centered on the canvas. So, goal is ultimate result, which you want to achieve after performing some specific tasks.

14.2 Donald Norman's Model

We have already studied Donald Norman's Model of interaction. In which user chooses a goal, formulate a plan of action, which is then executed at the computer interface. When the plan, or part of the plan has been executed, the user observes the computer interface to evaluate the result of the execution plan, and to determine further actions.

© Copyright Virtual University of Pakistan

The two major parts, execution and evaluation, of interactive cycle are further subdivided into seven stages, where each stage is an activity of the user. Seven stages of action are shown in figure. To understand these we see an example, which was also used by Norman.

Imagine you are sitting reading as evening falls. You decide you need more light; that is you establish the goal to get lighter. Form there you form an intention to switch on the desk lamp, and you specify the actions required to reach over and press the lamp switch. If some one else is closer, the intention may be different-you may ask them to switch on the light for you. Your goal is the same but the intention and actions are different. When you have executed the action you perceive the result, either the light is on or it isn't and you interpret this, based on your knowledge of the world. For example, if the light does not come on you may interpret this as indicating he bulb has blown or the lamp is not plugged into the mains, you will formulate the new state according to the original goals – is there is now enough light? It so, the cycle is completed. It not, you may formulate a new intention to switch on the main ceiling light as well.

Gulf of execution and evaluation

Norman also describes the two gulfs, which represent the problems that are caused by some interfaces to their users.

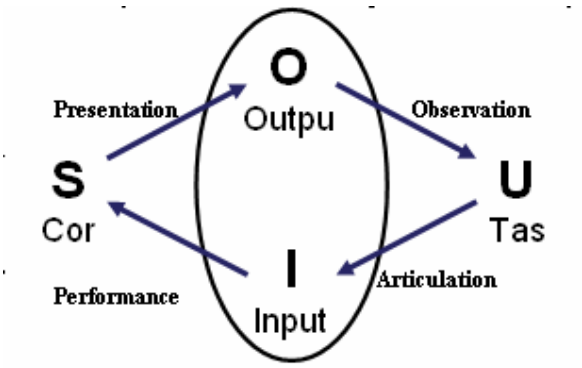
Gulf of execution

Gulf of execution is the difference between the user's formulation of the actions to reach the goal and the actions allowed by the system. If the action allowed by the system correspond to those intended by the user, the interaction will effective. The interface should therefore aim to reduce this gulf of execution.

Gulf of evaluation

The gulf of evaluation is the distance between the physical presentation of the system state and the expectation of the user. If the user can readily evaluate the presentation in terms of his goal, the gulf of evaluation is small. The more effort that is required on the part of the user to interpret the presentation, the less effective the interaction.

14.3 The interaction framework



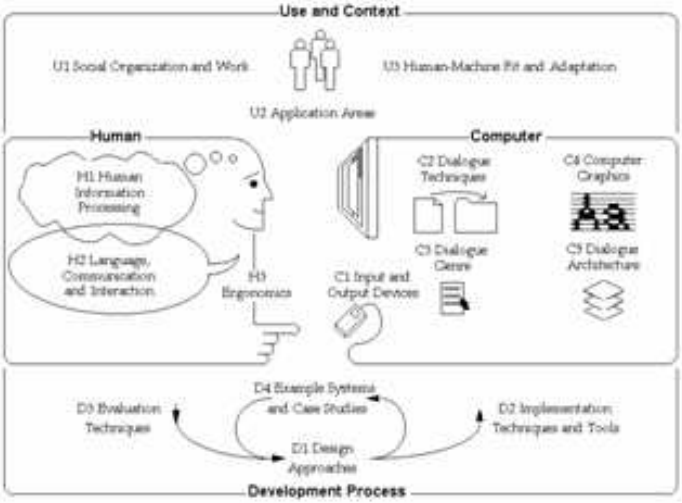
The interaction framework breaks the system into four main components as shown in figure. The nodes represent the four major components in an interactive system – the

© Copyright Virtual University of Pakistan

System, the User, the Input and the Output. Each component has its own language. The system and user are each described by means of a language that can express concepts relevant in the domain of the application. The system's language is referred as the core language and the user's language is referred as the task language. The core language describes computational attributes of the domain relevant to the system state, whereas the task language describes psychological attributes of the domain relevant to the user state. There are also languages for both the input and output components. Input and output together form the interface.

As the interface sits between the user and the system, there are four steps in the interactive cycle, each corresponding to a translation from one component to another, as shown by the labeled arcs in figure. The user begins the interactive cycle with the formulation of a goal and a task o achieves that goal. The only way the user can manipulate the machine is through the input, and so the task must be articulated within the input language, the input language is translated into the core language as operations to be performed by the system. The system then transforms itself as described by the operations; the execution phase of the cycle is complete and the evaluation phase now begins. The system is in a new state, which must now be communicated to the user. The current values of system attributes are rendered as concepts or features of the output. It is then up to the user to observe the output and assess the results of the interaction relative to the original goal, ending the evaluation phase and, hence, the interactive cycle. There are four main translations involved in the interaction: articulation, performance, presentation and observation. The user's formulation of the desired task to achieve some goal needs to be articulated in the input language. The tasks are responses of the user and they need to be translated to stimuli for the input. As pointed out above, this articulation is judged in terms of the coverage from tasks to input and the relative ease with which the translation can be accomplished. The task is phrased in terms of certain psychological attributes that highlight the important features of the domain for the user. If these psychological attributes map clearly onto the input language, then articulation of the task will be made much simpler.

14.4 Frameworks and HCI



© Copyright Virtual University of Pakistan

The ACM SIGCHI Curriculum Development Group presents a framework and uses it to place different areas that relate to HCI. As you can see in the figure, the field of ergonomics addresses issues on the user side of the interface, covering input and output, as well as the user's immediate context. Dialog design and interface styles can be placed particularly along the input branch of the framework, addressing both articulation and performance. However, dialog is most usually associated with the computer and so is biased to that side of the framework. Presentation and screen design relates to the output branch of the framework. The entire framework can be placed within a social and organizational context that also affects the interaction. Each of these areas has important implications for the design of interactive systems and the performance of the user.
Let us first take a brief look.

Ergonomics

Ergonomics (or human factors) is traditionally the study of the physical characteristic of the interaction: how the controls are designed, the physical environment in which the interaction takes place and the layout and physical qualities of the screen. A primary focus is on user performance and how the interface enhances or detracts from this. In seeking to evaluate these aspects of the interaction, ergonomics will certainly also touch upon human psychology and system constraints. It is a large and established field, which is closely related to but distinct from HCI.
Physical aspects of Interface are as follow:

- Arrangement of controls and displays
- The physical environment
- Health issues
- Use of colors

Arrangement of controls and displays

We already have discussed in previous lectures the perceptual and cognitive issues that affect the way we present information on a screen and provide control mechanisms to the user. In addition to these cognitive aspects of design, physical aspects are also important. The user should group sets of controls and parts of the display logically to allow rapid access. This may not seem so important when we are considering a single user of a spreadsheet on a PC, but it becomes vital we turn to safety-critical applications such as plant control, aviation and air traffic control. In each of these contexts, users are under pressure and are faced with a huge range of displays and controls. Here it is crucial that the physical layout of these be appropriate. Indeed, returning to the less critical PC application, inappropriate placement of controls and displays can lead to inefficiency and frustration.

Industrial Interface

The interfaces to office systems have changed dramatically since the 1980s. However, some care is needed in transferring the idioms of office-based systems into the industrial domain. Office information is primarily textual and slow varying, whereas industrial interfaces may require the rapid assimilation of multiple numeric displays; each of which is varying in response to the environment. Furthermore, the environment conditions may rule out certain interaction styles. Consequently, industrial interfaces raise some additional design issues rarely encountered in the office.

© Copyright Virtual University of Pakistan

Glass interfaces vs. dials and knobs

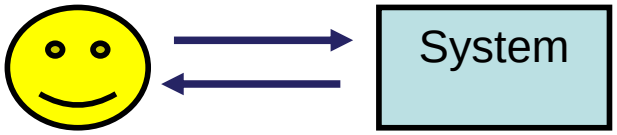
The traditional machine interface consists of dials and knobs directly wired or piped to the equipment. Increasingly, some or all of the controls are replaced with a glass interface, a computer screen through which the equipment is monitored and controlled. Many of the issues are similar for the two kinds of interface, but glass interfaces do have some special advantages and problems. For a complex system, a glass interface can be both cheaper and more flexible, and it is easy to show the same information in multiple forms.

Indirect manipulation

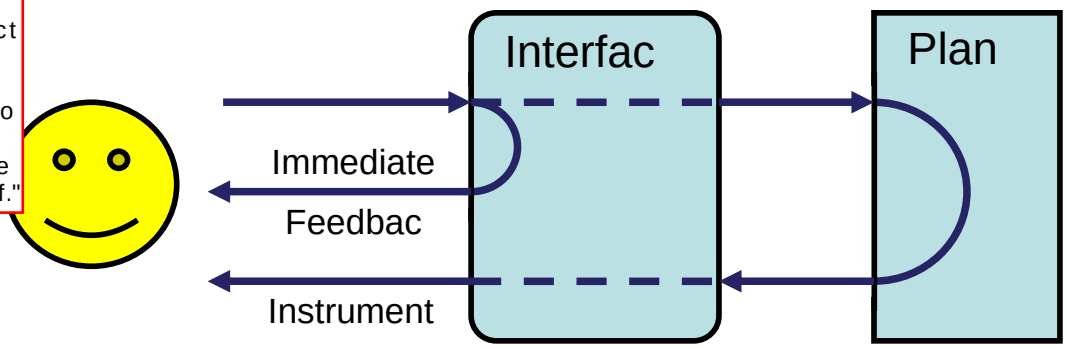
Direct Manipulation means the user directly controls and changes things in a computer system (like dragging a file into a folder).

Indirect manipulation is a type of interaction where the user controls or influences a system through an interface, but does not interact with the system or objects directly.
In other words:"Indirect manipulation means the user gives commands or inputs to a system, and the system carries out the actions on their behalf."

The phrase 'direct manipulation, dominates office system design as shown in figure (a). there are arguments about its meaning and appropriateness even there, but it is certainly dependent on the user being in primary control of the changes in the interface. The autonomous nature of industrial processes makes this an inappropriate model. In a direct manipulation system, the user interacts with an artificial would inside the computer.



In contrast, an industrial interface is merely an intermediary between the operator and the real world. One implication of this indirectness is that the interface must provide feedback at two levels as shown in figure (b). at one level, the user must receive immediate



feedback, generated by the interface, that keystrokes and other actions have been received. In addition, the user's action will have some effect on the equipment controlled by the interface and adequate monitoring must be provided for this.

The indirectness also causes problems with simple monitoring tasks. Delays due to periodic sampling, slow communication and digital processing often mean that the data displayed are somewhat out of date. If the operator is not aware of these delays, diagnoses of system state may be wrong. These problems are compounded if the interface produces summary information displays. If the data comprising such a display are of different timeliness the result may be misleading.

The physical environment of the interaction

As well as addressing physical issues in the layout and arrangement of the machine interface, ergonomics is concerned with the design of the work environment itself. Where will the system be used? By whom will it be used? Will users be sitting, standing or moving about? Again, this will depend largely on the domain and will be more critical in

© Copyright Virtual University of Pakistan

specific control and operational setting s than in general computer use. However, the physical environment in which the system is used may influence how
will it is accepted and even the health and safety f its users. It should therefore be considered in all design.

Health issues

Perhaps we do not immediately think of computer use as a hazardous activity but we should bear in mind possible consequences of our designs on the health and safety of users. Leaving aside the obvious safety risks of poorly designed safety-critical systems. There are a number of factors in that may affect the use of more general computers. Again these are factors in the physical environment that directly affect the quality of the interaction and the user's performance:

Physical position

As we discussed earlier users should be able to reach all controls comfortably and see all displays. Users should not be expected to stand for long periods and, if sitting, should be provided with back support.

Temperature

Although most users can adapt to slight changes in temperature without adverse effect, extremes of hot or cold will affect performance and, in excessive cases, health.

Lighting

The lighting level will again depend on the work environment. However, adequate lighting should be provided to allow users to see the computer screen without discomfort or eyestrain. The light source should also be positioned to avoid glare affecting the display.

Noise

Excessive noise can be harmful to health, causing the user pain, and in acute cases, loss of hearing. Noise level should be maintained at a comfortable level in the work environment.

Time

The time users spend using the system should also be controlled.

The use of color

Ergonomics has a close relationship to human psychology in that it is also concerned with the perceptual limitations of humans. For example, the use of color in displays is an ergonomics issue. The human visual system has some limitations with regard to color, including the number of colors that are distinguishable and the relatively low blue acuity. Color used in display should be as distinct as possible and the distinction should not be affected by changes in contrast. The colors used should also correspond to common conventions and user expectation. However, we should remember that color conventions are culturally determined.

© Copyright Virtual University of Pakistan

14.5 Interaction styles

Interaction is communication between computer and human (user). For a successful enjoyable communication interface style has its own importance.
There are a number of common interface styles including

- Command line interface
- Menus
- Natural language

Question/answer and query dialog

- Form fills and spreadsheets
- WIMP
- Point and click
- Three-dimensional interfaces.

Command line interface

Command line interface was the first interactive dialog style to be commonly used and, in spite of the availability of menu-driven interface, it is still widely used. It provides a means of expressing instructions to the computer directly, using some function keys, single characters, abbreviations or whole-word commands.
Command line interface are powerful in that they offer direct access to system functionality, and can be combined to apply a number of tools to the same data. They are also flexible: the command often has a number of options or parameters that will vary its behavior in some way, and it can be applied to many objects at once, making it useful for repetitive tasks.

Menu

In the menu-driven interface, the set of options available to the user is displayed on the screen and selected using the mouse, or numeric or alphabetic keys. Since the options are visible they are less demanding of the user, relying on recognition rather than recall. However, menu options still need to be meaningful and logically grouped to aid recognition. Often menus are hierarchically ordered and the option required is not available at the top layer of the hierarchy. The grouping and naming of menu options then provides the only cue for the user to find the required option. Such systems either can be purely text based, with the menu options being presented as numbered choices, or may have a graphical component in which the menu appears within a rectangular box and choices are made, perhaps by typing the initial letter of the desired selection, or by entering the associated number, or by moving around the menu with the arrow keys. This is restricted form of a full WIMP system.



© Copyright Virtual University of Pakistan

Natural Language

Perhaps the most attractive means of communicating with computers, at least at first glance, is by natural language. Users unable to remember a command or lost in a hierarchy of menus, may long for the computer that is able to understand instructions expressed in everyday words. Unfortunately, however, the ambiguity of natural language makes it very difficult for a machine to understand.

Question/answer and query dialog

Question and answer dialog is a simple mechanism for providing input to an application in a specific domain. The user is asked a series of questions and so is led through the interaction step by step. These interfaces are easy to learn and use, but are limited in functionality and power. As such, they are appropriate for restricted domains and for novice or casual users.

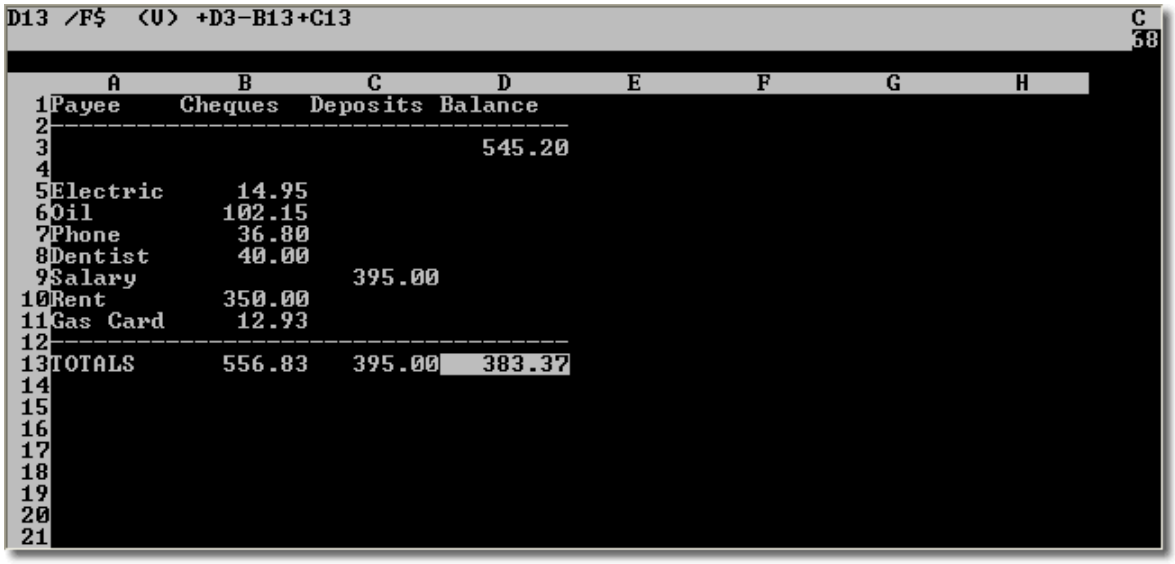
Query languages, on the other hand, are used to construct queries to retrieve information from a database. They use natural-language-style phrases, but in fact require specific syntax, as well as knowledge of database structure. Queries usually require the user to specify an attribute or attributes for which to search the database, as well as the attributes of interest to be displayed. This is straightforward where there is a single attribute, but becomes complex when multiple attributes are involved, particularly of the user is interested in attribute A or attribute B, or attribute A and not attribute B, or where values of attributes are to be compared. Most query language do not provide direct confirmation of what was requested, so that the only validation the user has is the result of the search. The effective use of query languages therefore requires some experience.

Form-fills and spreadsheets

Form-filling interfaces are used primarily for data entry but can be useful in data retrieval applications. The user is presented with a display resembling a paper form, with slots to fill in as shown in figure. Most form-filling interfaces allow easy movement around the form and allow some fields to be left blank. They also require correction facilities, as users may change their minds or make a mistake about the value that belongs in each field.

Spreadsheets are sophisticated variation of form filling. The spreadsheet comprises a grid of cells, each of which can contain a value or a formula. The formula can involve the value of other cells. Now days MS Excel is used widely. In past VISICALC and Lotus 123 had been used.

© Copyright Virtual University of Pakistan



VISICALC

The WIMP Interfaces

Currently many common environments for interactive computing are examples of the WIMP interface style, often simply called windowing systems. WIMP stands for windows, icons, menus, and pointers, and is default interface style for the majority of interactive computer systems in use today, especially in the PC and desktop workstation arena.

Point and Click interface

In most multimedia systems and in web browsers, virtually all actions take only a single click of the mouse button. You may point at a city on a map and when you click a window opens, showing you tourist information about the city. You may point at a word in some text and when you click you see a definition of the word. You may point at a recognizable iconic button and when you click some action is performed.

Three-dimensional interfaces

There is an increasing use of three-dimensional effects in user interfaces. The most obvious example is virtual reality, but VR is only part of a range of 3D techniques available to the interface designer.

The simplest technique is where ordinary WIMP elements, buttons, scroll bars, etc,, are given a 3D appearance using shading, giving the appearance of being sculpted out of stone.

© Copyright Virtual University of Pakistan

Lecture 15.

Interaction Paradigms

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Describe WIMP interfaces in detail
- Discuss different interaction paradigms

We have briefly discussed about the WIMP interfaces in last lecture. Today we will discuss WIMP interfaces in detail.

15.1 The WIMP Interfaces

In our last lecture we have already discussed the four key features of the WIMP interface that give it its name – windows, icons, pointers and menus – and today we will discuss these in greater detail. There are also many additional interaction objects and techniques commonly used in WIMP interfaces, some designed for specific purposes and others more general. Our discussion will cover the toolbars, menus, buttons, palettes and dialog boxes. Together, these elements of the WIMP interfaces are called widgets, and they comprise the toolkit for interaction between user and system.

Windows

Windows are areas of the screen that behave as if they were independent terminals in their own right. A window can usually contain text or graphics, and can be moved or resized. More than one window can be on a screen at once, allowing separate tasks to be visible at the same time. Users can direct their attention to the different windows as they switch from one thread of work to another.

If one window overlaps the other, the back window is partially obscured, and then refreshed when exposed again. Overlapping windows can cause problems by obscuring vital information, so windows may also be tiled, when they adjoin but do not overlap each other. Alternatively, windows may be placed in a cascading fashion, where each new window is placed slightly to the left and below the previous window. In some systems this layout policy is fixed, in others the user can select it.

Usually windows have various things associated with them that increase their usefulness. Scrollbars are one such attachment, allowing the user to move the contents of the window up and down, or from side to side. This makes the window behave as if it were a real window onto a much larger world, where new information is brought into view by manipulating the scrollbars.

There is usually a title bar attached to the top of a window, identifying it to the user, and there may be special boxes in the corners of the window to aid resizing, closing, or making as large as possible. Each of these can be seen in the figure.

© Copyright Virtual University of Pakistan

In addition, some systems allow windows within windows. For example, in Microsoft Office applications, such as Excel and Word, each application has its own window and then within this each document has a window. It is often possible to have different layout policies within the different application windows.

Icons

Windows can be closed and lost forever, or they can be shrunk to some much reduced representation. A small picture is used to represent a closed window, and this representation is known as an icon. By allowing icons, many windows can be available on the screen at the same time, ready to be expanded to their full size by clicking on the icon. Shrinking a window to its icon is known as iconifying the window. When a user temporarily does not want to follow a particular thread of dialog, he can suspend that dialog by iconifying the window containing the dialog. The icon saves space on the screen and serves as a remainder to the user that he can subsequently resume the dialog by opening up the window. Figure shows a few examples of icons used in a typical windowing system (Microsoft).

Icons can also be used to represent other aspects of the system, such as a wastebasket for throwing unwanted files into, or various disks, programs or functions, that are accessible to the user. Icon can take many forms: they can be realistic representation of the objects that they stand for, or they can be highly stylized. They can even be arbitrary symbols, but these can be difficult for users to interpret.

Pointers

The Pointer is an important component of the WIMP interface, since the interaction style required by WIMP relies very much on pointing and selecting things such as icons. The mouse provides an input device capable of such tasks, although joysticks and trackballs are other alternatives. The user is presented with a cursor on the screen that is controlled by the input device.

The different shapes of cursor are often used to distinguish modes, for example the normal pointer cursor maybe an arrow, but change to change to cross-hairs when drawing a line. Cursors are also used to tell the user about system activity, for example a watch or hourglass cursor may be displayed when the system s busy reading a file. Pointer cursors are like icons, being small bitmap images, but in addition all cursors have a hot-spot, the location to which they point.

Menus

The last main feature of the windowing system is the menu, an interaction technique that is common across many non-windowing systems as well. A menu presents a choice of operations or services that can be performed by the system at a given time. As we discussed our ability to recall information is inferior to our ability to recognize it from some visual cue. Menus provide information cues in the form of an ordered list of operations that can be scanned. This implies that the names used for the commands in the menu should be meaningful and informative.

The pointing device is used to indicate the desired option. As the pointer moves to the position of a menu item, the item is usually highlighted to indicate that it is the potential candidate for selection. Selection usually requires some additional user action, such as

© Copyright Virtual University of Pakistan

Keyboard Accelerators: Key combinations that perform the same actions as menu items, allowing faster interaction.

Buttons: Clickable areas on the screen that perform a specific action when pressed.

Radio Buttons: A group of buttons where only one option can be selected at a time.

Check Boxes: Toggle buttons that let you select multiple independent options.

Toolbars: A row or column of small icon-based buttons that provide quick access to commonly used functions.

Palettes: A visible collection of icons representing modes or tools, often used in graphics or editing software.

Dialog Boxes: Pop-up windows that provide information or ask the user for input during specific tasks.

pressing a button on the mouse that controls the pointer cursor on the screen or pressing some special key on the keyboard. Menus are inefficient when they have too many items, and so cascading menus are utilized, in which item selection opens up another menu adjacent to the item, allowing refinement of the selection. Several layers of cascading menus can be used.

The main menu can be visible to the user all the time, as a menu bar and submenus can be pulled down or across from it upon request. Menu bars are often placed at the top of the screen or at the top of each window. Alternative includes menu bars along one side of the screen, or even placed amongst the windows in the main 'desktop' area. Websites use a variety of menu bar locations, including top, bottom and either side of the screen. Alternatively, the main menu can be hidden and upon request it will pop up onto the screen. These pop-up menus are often used to present context-sensitive options, for example allowing one to examine properties of particular on-screen objects. In some systems they are also used to access more global actions when the mouse is depressed over the screen background.

Pull-down menus are dragged down from the title at the top of the screen, by moving the mouse pointer into the title par area and pressing the button. Fall-down menus are similar, except that the menu automatically appears when the mouse pointer enters the title bar, without the user having to press the button. Some menus explicitly asked to go away. Pop up menus appear when a particular region of the screen, may be designated by an icon, is selected, but they only stay as long as the mouse button is depressed.

Another approach to menu selection is to arrange the options in a circular fashion. The pointer appears in the center of the circle, and so there is the same distance to travel to any of the selections. This has the advantages that it is easier to select items, since they can each have a larger target area, and that the selection time for each item is the same, since the pointer is equidistant from them all. However, these pie menus take up more screen space and are therefore less common in interface.

The major problems with menus in general are deciding what items to include and how to group those items. Including too many items makes menus too long or creates too many of them, whereas grouping causes problems in that items that relate to the same topic need to come under the same heading, yet many items could be grouped under more than one heading. In pull-down menus the menu label should be chosen to reflect the function of the menu items, and items grouped within menus by function. These groupings should be consistent across applications so that the user can transfer learning to new applications. Menu items should be ordered in the menu according to importance and frequency of use, and appropriate functionalities should be kept apart to prevent accidental selection of the wrong function, with potentially disastrous consequences.

Keyboard accelerators

Menus often offer keyboard accelerators, key combinations that have the same effect as selecting the menu item. This allows more expert users, familiar with the system, to manipulate things without moving off the keyboard, which is often faster. The accelerators are often displayed alongside the menu item so that frequent use makes them familiar.

© Copyright Virtual University of Pakistan

Buttons

Buttons are individual and isolated regions within display that can be selected by the user to invoke specific operations. These regions are referred to as buttons because they are purposely made to resemble the push buttons you would find on a control panel. 'Pushing' the button invokes a command, the meaning of which is usually indicated by a textual label or a small icon.

Radio Buttons

Buttons can also be used to toggle between two states, displaying status information such as whether the current font is italicized or not in a word processor or selecting options on a web form. Such toggle buttons can be grouped together to allow a user to select one feature form a set of mutually exclusive options, such as the size in points of the current font. These are called radio buttons.

Check boxes

It a set of options is not mutually exclusive, such as font characteristics like bold, italic and underlining, and then a set of toggle buttons can be used to indicate the on/off status of the options. This type of collection of buttons is sometimes referred to as check boxes

Toolbars

Many systems have a collection of small buttons, each with icons, placed at the top or side of the window and offering commonly used functions. The function of this toolbar is similar to a menu bar, but as the icons are smaller than the equivalent text more functions can be simultaneously displayed. Sometimes the content of the toolbar is fixed, but often users can customize it, either changing which functions area made available, or choosing which of several predefined toolbars is displayed

Palettes

In many application programs, instructions can either one of several modes. The defining characteristic of modes is that the interpretation of actions, such as keystrokes or gestures with the mouse, changes as the mode change. For example, using the standard UNIX text editor vi, keystrokes can be interpreted either as operations to insert characters in the document or as operations to perform file manipulation. Problems occur if the user is not aware of the current mode. Palettes are a mechanism for making the set of possible modes and the active mode visible to the user. A palette is usually a collection of icons that are reminiscent of the purpose of the various modes. An example in a drawing package would be a collection of icons to indicate the pixel color or pattern that is used to fill in objects, much like an artist's palette for paint.

Some systems allow the user to create palettes from menus or toolbars. In the case of pull-down menus, the user may be able 'tear off' the menu, turning it into a palette showing the menu items. In the case of toolbars, he may be able to drag the toolbar away from its normal position and place it anywhere on the screen. Tear-off menus are usually those that are heavily graphical anyway, for example line style of color selection in a drawing package.

© Copyright Virtual University of Pakistan

Dialog boxes

Dialog boxes are information windows used by the system to bring the user's attention to some important information, possibly an error or a warning used to prevent a possible error. Alternatively, they are used to invoke a sub dialog between user and system for a very specific task that will normally be embedded within some larger task. For example, most interactive applications result in the user creating some file that will have to be named and stored within the filing system. When the user or the file and indicate where it is to be located within the filing system. When the save sub dialog is complete, the dialog box will disappear. Just as windows are used to separate the different threads of user-system dialog, so too are dialog boxes used to factor out auxiliary task threads from the main task dialog.

15.2 Interaction Paradigms

We believe that we now build interactive systems that are more usable than those built in the past. We also believe that there is considerable room for improvement in designing more usable systems in the future. The great advances in computer technology have increased the power of machines and enhanced the bandwidth of communication between human and computer. The impact of the technology alone, however, is not sufficient to enhance its usability. As our machines have become more powerful, they key to increased usability has come from the creative and considered application of the technology to accommodate and augment the power of the human. Paradigms for interaction have for the most part been dependent upon technological advances and their creative application to enhance interaction.

By interaction paradigm, it is meant a particular philosophy or way of thinking about interaction design. It is intended to orient designers to the kinds of questions they need to ask. For many years the prevailing paradigm in interaction design was to develop application for the desktop – intended to be used by single user sitting in front of a CPU, monitor, keyboard and mouse. A dominant part of this approach was to design software applications that would run using a GUI or WIMP interface.

Recent trend has been to promote paradigms that move beyond the desktop. With the advent of wireless, mobile, and handheld technologies, developers started designing applications that could be used in a diversity of ways besides running only on an individual's desktop machine.

Time sharing

In the 1940s and 1950s, the significant advances in computing consisted of new hardware technologies. Mechanical relays were replaced by vacuum electron tubes. Tubes were replaced by transistors, and transistors by integrated chips, all of which meant that the amount of sheer computing power was increasing by orders of magnitude. By the 1960s it was becoming apparent that the explosion of growth in computing power would be wasted if there were not an equivalent explosion of ideas about how to channel that power. One of the leading advocates of research into human-centered applications of computer technology was J.C.R Licklider, who became the director of the Information Processing Techniques Office of the US Department of Defense's Advanced Research Agency (ARPA). It was Licklider's goal to finance various research centers across the United States in order to encourage new ideas about how best to apply the burgeoning computing technology.

© Copyright Virtual University of Pakistan

One of the major contributions to come out of this new emphasis in research was the concept of time-sharing, in which a single computer could support multiple users. Previously, the human was restricted to batch sessions, in which complete jobs were submitted on punched cards or paper tape to an operator who would then run them individually on the computer. Time-sharing systems of the 1960s made programming a truly interactive venture and brought about a subculture of programmers known as 'hackers' – single-minded masters of detail who took pleasure in understanding complexity. Though the purpose of the first interactive time-sharing systems was simply to augment the programming capabilities of the early hackers, it marked a significant stage in computer applications for human use.

Video display units

As early as the mid-1950s researchers were experimenting with the possibility of presenting and manipulating information from a computer in the form of images on a video display unit (VDU). These display screens could provide a more suitable medium than a paper printout for presenting vast quantities of strategic information for rapid assimilation. It was not until 1962, however, when a young graduate student at the Massachusetts Institute of Technology (MIT), Ivan Sutherland, astonished the established computer science community with the Sketchpad program, that the capabilities of visual images were realized.

Sketchpad demonstrated two important ideas. First, computers could be used for more than just data processing. They could extend the user's ability to abstract away from some levels of detail, visualizing and manipulating different representations of the same information. Those abstractions did not have to be limited to representations in terms of bit sequences deep within the recesses of computer memory. Rather, the abstraction could be make truly visual. To enhance human interaction, the information within the computer was made more amenable to human consumption. The computer was made to speak a more human language, instead of the human being forced to speak more like a computer. Secondly, Sutherland's efforts demonstrated how important the contribution of one creative mind could be to the entire history of computing.

Programming toolkits

Dougles Engelbart's ambition since the early 1950s was to use computer technology as a means of complementing human problem-solving activity. Engelbart's idea as a graduate student at the University f California at Berkeley was to use the computer to teach humans. This dream of naïve human users actually learning from a computer was a stark contrast to the prevailing attitude of his contemporaries that computers were purposely complex technology that only the intellectually privileged were capable of manipulating.

Personal computing

Programming toolkits provide a means for those with substantial computing skills to increase their productivity greatly. But Engelbart's vision was not exclusive to the computer literate. The decade of the 1970s saw the emergence of computing power aimed at the masses, computer literate or not. One of the first demonstrations that the powerful tools of the hacker could be made accessible to the computer novice was a graphics programming language for children called LOGO. The inventor, Seymen Papert, wanted to develop a language that was easy for children to use. He and his colleagues from MIT and elsewhere designed a computer-controlled mechanical turtle that dragged a pen along

© Copyright Virtual University of Pakistan

a surface to trace its path. In the early 1970s Alan Kay view of the future of computing was embodied in small, powerful machines, which were dedicated to single users, that is personal computers. Together with the founding team of researchers at the Xerox Palo Alto Research Center, Kay worked on incorporating a powerful and simple visually based programming environment, Smalltalk, for the personal computing hardware that was just becoming feasible. As technology progresses, it is now becoming more difficult to distinguish between what constitutes a personal computer, or workstation, and what constitutes a mainframe.

Window systems and the WIMP interface

With the advent and immense commercial success of personal computing, the emphasis for increasing the usability of computing technology focused on addressing the single user who engaged in a dialog with the computer in order to complete some work. Humans are able to think about more than one thing at a time, and in accomplishing some piece of work, they frequently interrupt their current train of thought to pursue some other related piece of work. A personal computer system which forces the user to progress in order through all of the tasks needed to achieve some objective, from beginning to end without any diversions, does not correspond to that standard working pattern. If the personal computer is to be an effective dialog partner, to must be as flexible in its ability to change the topic as the human is.

But the ability to address the needs of a different user task is not the only requirement. Computer systems for the most part react to stimuli provided by the user, so they are quite amenable to a wandering dialog initiated by the user. As the ser engages in more than one plan of activity over a stretch of time, it becomes difficult for him to maintain the status of the overlapping threads of activity.

Interaction based on windows, icons, menus, and pointers—the WIMP interface—is now commonplace. These interaction devices first appeared in the commercial marketplace in April 1981, when Xerox Corporation introduced the 8010 Star Information System.

The metaphor

Metaphor is used quite successfully to teach new concepts in terms of ones, which are already understood. It is no surprise that this general teaching mechanism has been successful in introducing computer novices to relatively foreign interaction techniques. Metaphor is used to describe the functionality of many interaction widgets, such as windows, menus, buttons and palettes. Tremendous commercial successes in computing have arisen directly from a judicious choice of metaphor. The Xerox Alto and Star were the first workstations based on the metaphor of the office desktop. The majority of the management tasks on a standard workstation have to do with the file manipulation. Linking the set of tasks associated with file manipulation to the filing tasks in a typical office environment makes the actual computerized tasks easier to understand at first. The success of the desktop metaphor is unquestionable. Another good example in the personal computing domain is the widespread use of the spreadsheet for accounting and financial modeling.

Very few will debate the value of a good metaphor for increasing the initial familiarity between user and computer application. The danger of a metaphor is usually realized after the initial honeymoon period. When word processors were first introduced, they relied

© Copyright Virtual University of Pakistan

heavily on the typewriter metaphor. The keyboard of a computer closely resembles that of a standard typewriter, so it seems like a good metaphor from any typewriter. For example, the space key on a typewriter is passive, producing nothing on the piece of paper and just moving the guide further along the current line. For a typewriter, a space is not a character. However, for a word processor, the blank space is a character, which much be inserted within a text just as any other character is inserted. So an experienced typist is not going to be able to predict his experience with a preliminary understanding of a word processor.

Another problem with a metaphor is the cultural bias that it portrays. With the growing internationalization of software, it should not be assumed that a metaphor will apply across national boundaries. A meaningless metaphor will only add another layer of complexity between the user and the system.

Direct Manipulation

In the early 1980s as the price of fast and high-quality graphics hardware was steadily decreasing, designers were beginning to see that their products were gaining popularity as their visual content increased. As long as the user-system command line prompt computing was going to stay within the minority population of the hackers who reveled in the challenge of complexity. In a standard command line interface, the only way to get any feedback on the results of previous interaction is to know that you only have to ask for it and to know how to ask for it. Rapid visual and audio feedback on a high-resolution display screen or through a high-quality sound system makes it possible to provide evaluative information for every executed user action.

Rapid feedback is just one feature of the interaction technique known as direct manipulation. Ben Shneiderman is attributed with coining this phrase in 1982 to describe the appeal of graphics-based interactive systems such as Sketchpad and the Xerox Alto and Star. He highlights the following features of a direct manipulation interface.

- Visibility of the objects of interest
- Incremental action at the interface with rapid feedback on all actions
- Reversibility of all actions, so that users are encouraged to explore without severe penalties
- Syntactic correctness of all actions, so that every user action is a legal operation
- Replacement of complex command language with actions to manipulate directly the visible objects.

The first real commercial success which demonstrated the inherent usability of direct manipulation interfaces for the general public was the Macintosh personal computer, introduced by Apple Computer, Inc. in 1984 after the relatively unsuccessful marketing attempt in the business community of the similar but more pricey Lisa computer. The direct manipulation interface for the desktop metaphor requires that the documents and folders are made visible to the user as icons, which represent the underlying files and directories. An operation such as moving a file from one directory to another is mirrored as an action on the visible document, which is picked and dragged along the desktop from one folder to the next.

© Copyright Virtual University of Pakistan

Language versus action

Whereas it is true that direct manipulation interface make some tasks easier to perform correctly, it is equally true that some tasks are more difficult, if not impossible. Contrary to popular wisdom, it is not generally true that action speak louder than words. The image, projected for direct manipulation was of the interface as a replacement for the underlying system as the world of interest to the user. Actions performed at the interface replace any need to understand their meaning at any deeper, system level. Another image is of the interface s the interlocutor or mediator between the user and the system. The user gives the interface instructions and it is then the responsibility of the interface to see that those instructions are carried out. The user-system communication is by means of indirect language instead of direct actions.

We can attach two meaningful interpretations to this language paradigm. The first requires that the user understands how the underlying system functions and the interface as interlocutor need not perform much translation. In fact, this interpretation of the language paradigm is similar to the kind of interaction, which existed before direct manipulation interfaces were around. In a way, we have come full circle.

The second interpretation does not require the user to understand the underlying system's structure. The interface serve a more active role, as it must interpret between the intended operation as requested by the user and the possible system operations that must be invoked to satisfy that intent. Because it is more active, some people refer to the interface as an agent in these circumstances. This kind of language paradigm can be seen in some internal system database, but you would not know how that information is organized.

Whatever interpretation is attached to the language paradigm, it is clear that it has advantages and disadvantages when compared with the action paradigm implied by direct manipulation interfaces. In the action paradigm, it is often much easier to perform simple tasks without risk o certain classes or error. For example, recognizing and pointing to an object reduces the difficulty of identification and the possibility of misidentification. On the other hand, more complicated tasks are often rather tedious to perform in the action paradigm, as they require repeated execution of the same procedure with only minor modification. In the language paradigm, there is the possibility describing a generic procedure once and then leaving it to be executed without further user intervention.

The action and language paradigms need not be completely separate. In the above example two different paradigms are distinguished by saying that generic and repeatable procedures can be described in the language paradigm and not in the action paradigm. An interesting combination of the two occurs in programming by example when a user can perform some routine tasks in the action paradigm and the system records this as a generic procedure. In a sense, the system is interpreting the user's actions as a language script that it can then follow.

Hypertext

In 1945, Vannevar Bush, then the highest-ranking scientific administrator in the US war effort, published an article entitled 'As We May Think' in The Atlantic Monthly. Bush was in charge of over 6000 scientists who had greatly pushed back the frontiers of scientific knowledge during the Second World War. He recognized that a major drawback of these prolific research efforts was that it was becoming increasingly difficult to keep in

© Copyright Virtual University of Pakistan

touch with the growing body of scientific knowledge in the literature. In his opinion, the greatest advantages of this scientific revolution were to be gained by those individuals who were able to keep abreast of an ever-increasing flow of information. To that end, he described an innovative and futuristic information storage and retrieval apparatus – the memex –, which was constructed with technology wholly existing in 1945 and aimed at increasing the human capacity to store and retrieve, connected pieces of knowledge by mimicking our ability to create random associative links.

An unsuccessful attempt to create a machine language equivalent of the memex on early 1960s computer hardware led Nelson on a lifelong quest to produce Xanadu, a potentially revolutionary worldwide publishing and information retrieval system based on the idea of interconnected, non-linear text and other media forms. A traditional paper is read from beginning to end, in a linear fashion. But within that text, there are often ideas or footnotes that urge the reader to digress into richer topic. The linear format for information does not provide much support for this random and associated browsing task. What Bush's memex suggested was to preserve the non-linear browsing structure in the actual documentation? Nelson coined the phrase hypertext in the mid 1960s to reflect this non-linear text structure.

Multi-modality

Multi-modality is the use of multiple human communication channels (like voice, touch, and vision) simultaneously for interacting with a system.

The majority of interactive systems still uses the traditional keyboard and a pointing device, such as a mouse, for input and is restricted to a color display screen with some sound capabilities for output. Each of these input and output devices can be considered as communication channels for the system and they correspond to certain human communication channels. A multi-modal interactive system is a system that relies on the use of multiple human communication channels. Each different channel for the user is referred to as a modality of interaction. In this sense, all interactive systems can be considered multi-model, for human have always used their visual and haptic channels in manipulating a computer. In fact, we often use our audio channel to hear whether the computer is actually running properly.

However, genuine multi-modal systems rely to an extent on simultaneous use of multiple communication channels for both input and output. Humans quite naturally process information by simultaneous use of different channels.

Computer-supported cooperative work

Another development in computing in the 1960s was the establishment of the first computer networks, which allowed communication between separate machines. Personal computing was all about providing individuals with enough computing power so that they were liberated from dumb terminals, which operated on time-sharing systems. It is interesting to note that as computer networks become widespread, individuals retained their powerful workstations but now wanted to reconnect themselves to the rest of the workstations in their immediate working environment, and even throughout the world. One result of this reconnection was the emergence of collaboration between individuals via the computer – called computer – supported cooperative work, or CSCW.

The main distinction between CSCW systems and interactive systems designed for a single user is that designer can no longer neglect the society within which any single user operates. CSCW systems are built to allow interaction between humans via the computer

© Copyright Virtual University of Pakistan

and so the needs of the many must be represented in the one product. A fine example of a CSCW system is electronic mail – email – yet another metaphor by which individuals at physically separate locations can communicate via electronic messages that work in a similar way to conventional postal systems.

The World Wide Web

Probably the most significant recent development interactive computing is the World Wide Web, often referred to as just the web, or WWW. The web is built on top of the Internet, and offers an easy to use, predominantly graphical interface to information, hiding the underlying complexities of transmission protocols, addresses and remote access to data.

The Internet is simply a collection of computers, each linked by any sort of data connections, whether it be slow telephone line and modem or high-bandwidth optical connection. The computers of the Internet all communicate using common data transmission protocols and addressing systems. This makes it possible for anyone to read anything from anywhere, in theory, if it conforms to the protocol. The web builds on this with its own layer of network protocol, a standard markup notation for laying out pages of information and a global naming scheme. Web pages can contain text, color images, movies, sound and, most important, hypertext links to other web pages. Hypermedia documents can therefore be published by anyone who has access to a computer connected to the Internet.

Ubiquitous computing

In the late 1980s, a group of researchers at Xerox PARC led by Mark Weiser, initiated a research program with the goal of moving human-computer interaction away from the desktop and out into our everyday lives. Weiser observed.

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. These words have inspired a new generation of researchers in the area of ubiquitous computing. Another popular term for this emerging paradigm is pervasive computing, first coined by IBM. The intention is to create a computing infrastructure that permeates our physical environment so much that we do not notice the computer may longer. A good analogy for the vision of ubiquitous computing is the electric motor. When the electric motor was first introduced, it was large, loud and very noticeable. Today, the average household contains so many electric motors that we hardly ever notice them anymore. Their utility led to ubiquity and, hence, invisibility.

Sensor-based and context-aware interaction

The yard-scale, foot-scale and inch-scale computers are all still clearly embodied devices with which we interact, whether or not we consider them ‘computers’. There are an increasing number of proposed and existing technologies that embed computation even deeper, but unobtrusively, into day-to-day life. Weiser’s dream was computers anymore’, and the term ubiquitous computing encompasses a wide range from mobile devices to more pervasive environments.

© Copyright Virtual University of Pakistan

Lecture 16.

HCI Process and Models

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the need of new software development models
- Understand the importance of user experience and usability in design

It has been said, "to err is human; to really screw up, you need a computer."
Inefficient mechanical systems can waste couple cents on every widget you build, but you can lose your entire company to bad information processes. The leverage that software-based products---and the engineers that build them---have on your company is enormous.
Sadly, our digital tools are extremely hard to learn, use, and understand, and they often cause us to fall short of our goals. This wastes money, time, and opportunity. As a business-savvy technologist/ technology-savvy businessperson, you produce software-based products or consume them---probably both. Having better, easier-to-learn, easier-to-use high-tech products is in your personal and professional best interest. Better products don't take longer to create, nor do they cost more to build. The irony is that they don't have to be difficult, but are so only because our process for making them is old-fashioned and needs fixing. Only long-standing traditions rooted in misconceptions keep us from having better products in today.

Consider a scenario: a website is developed of commerce system. The site is aesthetically very beautiful, technically it has no flaw and it has wonderful animated content on it. But if user is unable to find its desired information about the products or even he is unable to find the product out of thousands of products, so what of it's use. It is useless from the business point of view.

Here are some facts and figures:

Users can only find information 42% of the time
– Jared Spool
62% of web shoppers give up looking for the item they want to buy online
– Zona Research

50% of the potential sales from a site are lost because people cannot find the item they are

looking for

– Forrester Research
40% of the users who do not return to a site do so because their first visit resulted in a negative experience
– Forrester Research

© Copyright Virtual University of Pakistan

80% of software lifecycle costs occur after the product is released, in the maintenance phase - of that work, 80 % is due to unmet or unforeseen user requirements; only 20 % is due to bugs or reliability problems.
- IEEE Software

Around 63% of software projects exceed their cost estimates. The top four reasons for this are:

- Frequent requests for changes from users
- Overlooked tasks
- Users' lack of understanding of their own requirements
- Insufficient user-analyst communication and understanding

- Communications of the ACM
BOO.com, a $204m startup fails

– BBC News
Poor commercial web sites will kill 80% of Fortune 500 companies within a decade
Jakob Nielsen
So all above given facts reveals that the product with the bad user experience deserve to die!

	Scenario A	Scenario B
Revenue Potential	$100m	$100m
User Experience	Good	Bad
Sales Lost	0%	50%
Revenue Lost	$0m	$50m
Actual Revenue	**$100m**	**$50m**

All you can see that the real importance is of good interaction design. In above example you saw that whole the systems which were developed technically very well but failed just because of bad user experience, bad usability.

So, what if we design a system with good usability and good user experience. The result will be satisfaction and happiness. People will be happy with your product and they will buy your product, not only they will buy but also they will recommend others for your

© Copyright Virtual University of Pakistan

product. And other people will also buy your product and this will result a chain reaction. So, as you saw how your product can be successful if it provides a good user experience. Our discussion has a simple premise: if achieving the user's goal is the basis of our design process, the user will be satisfied and happy. If the user is happy, he will gladly pay us money (and recommend that others do the same), and then we will be successful as a business.

On the surface, this premise sounds quite obvious and straightforward: make the user happy, and your products will be a success. Why then are so many digital products so difficult and unpleasant to use? Why aren't we all either happy or successful—or both? Most digital products today emerge form the development process like a monster emerging from a bubbling tank. Developers, instead of planning and executing with their users in mind, end up creating technological solutions over which they ultimately have little control. Like a mad scientists, they fail because they have not imbued their creations with humanity. We are mainly concerned with software, so, as we have stated earlier the definition of software quality:

The extent to which a software product exhibits these characteristics

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

As you can see there are many characteristics to measure the quality of software and the usability is one of them. In fact, we can understand the nature of usability in two aspects

- Strategic
- Tactical

Strategic aspect guides us to think about user interface idioms – in other words, the way in which the user and the idiom interact. Tactical aspects give us hints and tips about using and creating user interface idioms, like dialog boxes and pushbuttons.

Integrating the strategic and tactical approaches is the key to designing effective user interaction and interface. For example, there is no such thing as an objectively good dialog box—the quality depends on the situation: who the user is and what his background and goals are. Merely applying a set of tactical dictums makes user interface creation easier, but it doesn't make the end result better. Deep thoughts about how users should interact with your system won't improve the software, either. What does work is maintaining a strategic sensitivity for how users actually interact with specific software and having at your command a tactical toolbox to apply in any particular situation.

© Copyright Virtual University of Pakistan

Ignorance about users

It is a sad truth that the digital technology industry doesn't have a good understanding of what it takes to make users happy. In fact, most technology products get built without much understanding of the users. We might know what market segment our users are in., how much money they make, how much money they like to spend on weekends, and what sort of cars they buy. May be we even having a vague idea what kind of jobs they have and some of the major tasks that they regularly perform? But does any of this tell us how to make them happy? Does it tell us how they will actually use the product we are building? Does it tell us why they are doing whatever it is they might need our product for, why they might want to choose our product over our competitors, or how we can make sure they do? Unfortunately, it does not.

Conflict of Interest

A second problem affects the ability of vendors and manufacturers to make users happy. There is an important conflict of interest in the world of digital product development: the people who build the products—programmers—are usually also the people who design them. Programmers are often required to choose between ease of coding and ease of use. Because programmers' performance is typically judged by their ability to code efficiently and meet incredibly tight deadlines, it isn't difficult to figure out what direction most software-enabled products take. Just as we would never permit the prosecutor in a legal trial to also adjudicate the case, we should make sure that those designing a product are not the same people building it. It simply isn't possible for a programmer to advocate for the user, the business, and the technology at the same time.

How can you achieve success?

The third reason that digital technology industry isn't cranking out successful products is that it has no reliable process for doing so. Or, to be more accurate, it doesn't have a complete process for doing so. Engineering departments follow—or should follow—rigorous engineering methods that ensure the feasibility and quality of the technology. Similarly, marketing, sales, and other business units follow their own well-established methods for ensuring the commercial viability of new products. What's left out is a repeatable, analytical process for transforming and understanding of users into products that both meet their needs and excite their imaginations.

While thinking about complex mechanical devices, we take for granted that they have been carefully designed for use, in addition to being engineered. Most manufactured objects are quite simple, and even complex mechanical products are quite simple when compared to most software and software-enabled products that can sport in excess of one million lines of code. Yet more software have never been undergone a rigorous design process from a user-centered perspective. The current process of determining what software will do and how it will communicate with the user is today closely intertwined with its construction. Programmers, deep in their thoughts of algorithms and code, "design" user interface the same way that miners "design" the landscape with their cavernous pits and enormous tailing piles. The software interface design process alternates between the accidental and the non-existent.

Many programmers today embrace the notion that integrating users directly into the programming process on a frequent bias—weekly or sometimes even daily—can solve design problems. Although this has the salutary effect of sharing the responsibility for

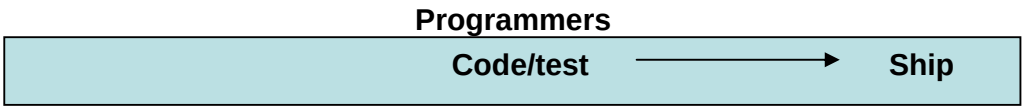
© Copyright Virtual University of Pakistan

design with the user, it ignores a serious methodological flaw: a confusion of domain knowledge with design knowledge. Users although they might be able to articulate the problems with an interaction, are not often capable of visualizing the solutions to those problems. Design is a specialized skill, just like programming. Programmer would never ask users to help them code; design problems should be treated no differently.

Evolution of Software Development Process

Originally programmers did it all

In the early days of software industry smart programmers dreamed up useful software, wrote and even tested it on their own. But as their businesses grew, the software business and software business and software products became more complicated.



Managers brought order

Inevitably, professional managers were brought in. Good product managers understand the market and competitors. They define software product by creating requirements documents. Often, however, requirements are little more than a list of features and managers find themselves having to give up features in order to meet schedule.



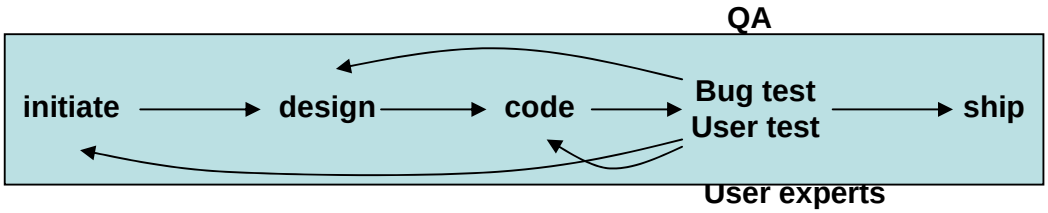
Testing and design became separate steps

As the industry matured, testing became a separate discipline and separate step in the process. In the move from command-line to graphical user interface, design and usability also became involved in the process, though often only at the end, and often only affecting visual presentation. Today common practice includes simultaneous coding and design followed by bug and user testing and then revision.

Design must precede the programming effort

A goal directed design approach to software development means that all decisions proceed from a format definition of the user and his or her goals. Definition of the user and user goals is the responsibility of the designer, thus design must precede programming.

© Copyright Virtual University of Pakistan



Design

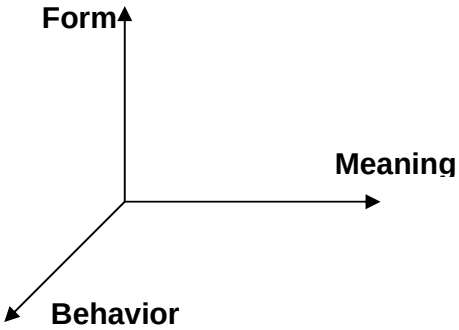
Design, according to industrial designer Victor Papanek, is the conscious and intuitive effort to impose meaningful order. Cooper proposes a somewhat detailed definition:

- Understanding the user's wants, needs, motivations, and contexts
- Understanding business, technical, and domain requirements and constraints
- Translating this knowledge into plans for artifacts whose form content, and behavior is useful, usable and desirable, as well as economically viable and technically feasible.

This definition applies across all design disciples, although the precise focus on form versus content versus behavior varies by design discipline.
When performed using the appropriate methods, design can provide the missing human connection in technological product. But clearly, the currently approach to the design of digital products either isn't working or isn't happening as advertised.

Three dimensions of designs

Interaction design focuses on an area that traditional design disciplines do not often explore: the design of behavior.
All design affects human behavior: architecture is about how people use spaces as much as it is about form and light. And what would be the point of a poster if no one acted on the information it presented?
However, it is only with the introduction of interactive technologies – courtesy of the computer – that the design of the behavior of artifacts and how this behavior affects and supports human goals and desires, has become a discipline worthy of attention.
One way of making sense of he difference is focus between interaction design and more traditional design is through a historical lens. In the first half of the twentieth century, designers focused primarily on form.



Three Dimensions of Design

Form – How something looks (its appearance and structure).

Meaning – What it communicates to users (the message or content).

Behavior – How it works and responds when a user interacts with it (the actions and interactions).

Later designers became increasingly concerned with meaning; for example, product designers and architects introduced vernacular and retro forms in the 70s. The trend continues today with retro-styled automobiles such as the PT Cruiser. Today, information designers continue the focus on meaning to include the design of usable content.

© Copyright Virtual University of Pakistan

Within the last fifteen years a growing group of designers have begun to talk about behavior: the dynamic ways that software-enabled products interact directly with users.
These concerns (form, meaning and behavior) are not exclusive. Interactive products must have each in some measure; software applications focus more on behavior and form, with less demand on content; web sites and kiosks focus more on content and form, with less sophisticated behaviors.

Definition interaction design

Simply put, interaction design is the definition and design of the behavior of artifacts, environment, and systems, as well as the formal elements that communicate that behavior. Unlike, traditional design disciplines, whose focus has historically been on form and, more recently, on content and meaning, interaction design seeks first to plan and describe how things behave and then, as necessary, to describe the most effective form to communicate those behaviors.

© Copyright Virtual University of Pakistan

Lecture 17.

HCI Process and Methodologies

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Describe the advantages and disadvantages of different Software Development Lifecycles
- In our last lecture we ended with the definition of design, today we will look at different design models used for development of software and we also consider their flaws. We start from where we left in last lecture, design.

The term design, even used in the context of designing a computer system, can have different meanings. In a review of general design philosophy, Jones (1981) found a number of definitions including:

> 'Finding the right physical components of a physical structure.'
> 'A goal-directed problem-solving activity.'
> Simulating what we want to make (or do) before we make (or do) it as many times as may be necessary to feel confident in the final result.'
> 'The imaginative jump from present facts to future possibilities.'
> 'A creative activity – it involves bringing into being something new and useful that has not existed previously.'

These descriptions focus on the process of design in general. On the topic of engineering design, Jones found this: 'Engineering design is the use of scientific principles, technical information and imagination in the definition of a mechanical structure, machine or system to perform pre-specified functions with the maximum economy and efficiency.' Webster, on the other hand, stresses the relationship between design representation and design process: 'A design is an information base that describes elaborations of representations, such as adding more information or even backtracking and exploring alternatives' (Webster, 1988).

Thus, 'design' refers to both the process of developing a product, artifact or system and to the various representations (simulations or models) of the product that are many produced during the design process. There are many representations, which are more or less appropriate in different circumstances. Designers need not only to be able to understand user' requirements, but also to be able to represent this understanding in different ways at different stages of the design. Selecting suitable representations is, therefore, important for exploring, testing, recording and communicating design ideas and decisions, both within the design team and with users.

In order to develop any product, two major activities have to be undertaken: the designer must understand the requirements of the product, and must develop the product.

© Copyright Virtual University of Pakistan

Understanding requirements involves looking at similar products, discussing the needs of the people who will use the product, and analyzing any existing systems to discover the problems with current designs. Development may include producing a variety of representations until a suitable artifact is produced.

17.1 Lifecycle models

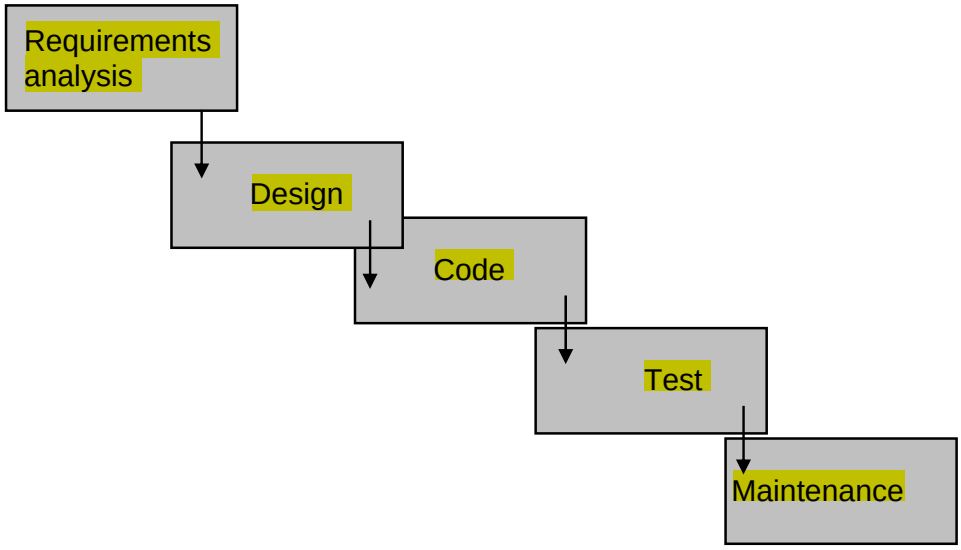
Understanding what activities are involved in interaction design is the first step to being able to do it, but it is also important to consider how the activities are related to one another so that the full development process can be seen. The term lifecycle model is used to represent a model that captures a set of activities and how they are related. Sophisticated models also incorporate a description of when and how to move from one activity to the next and a description of the deliverables for each activity. The reason such models are popular is that they allow developers, and particularly managers, to get an overall view of the development effort so that progress can be tracked, deliverables specified, resources allocated, targets set, and so on.

Existing models have varying levels of sophistication and complexity. For projects involving only a few experienced developers, a simple process would probably be adequate. However, for larger systems involving tens or hundreds of developers with hundreds or thousands of users, a simple process just isn't enough to provide the management structure and discipline necessary to engineer a usable product. So something is needed that will provide more formality and more discipline. Note that this does not mean that innovation is lost or that creativity is stifled. It just means that structured process is used to provide a more stable framework for creativity.

The waterfall model is a traditional software development approach where each step flows linearly into the next, like a waterfall.

The design of software systems

The traditional view of software engineering characterizes the development of software as consisting of number of processes and representations that are produced in an essentially linear fashion. This is often called waterfall model, because the output of each process 'tumbles down' neatly to the next activity, as illustrated in Figure.



The waterfall lifecycle was the first model generally known in software engineering and forms the basis of many lifecycle in use today. This is basically a linear model in which each step must be completed before the next step can be started. For example,

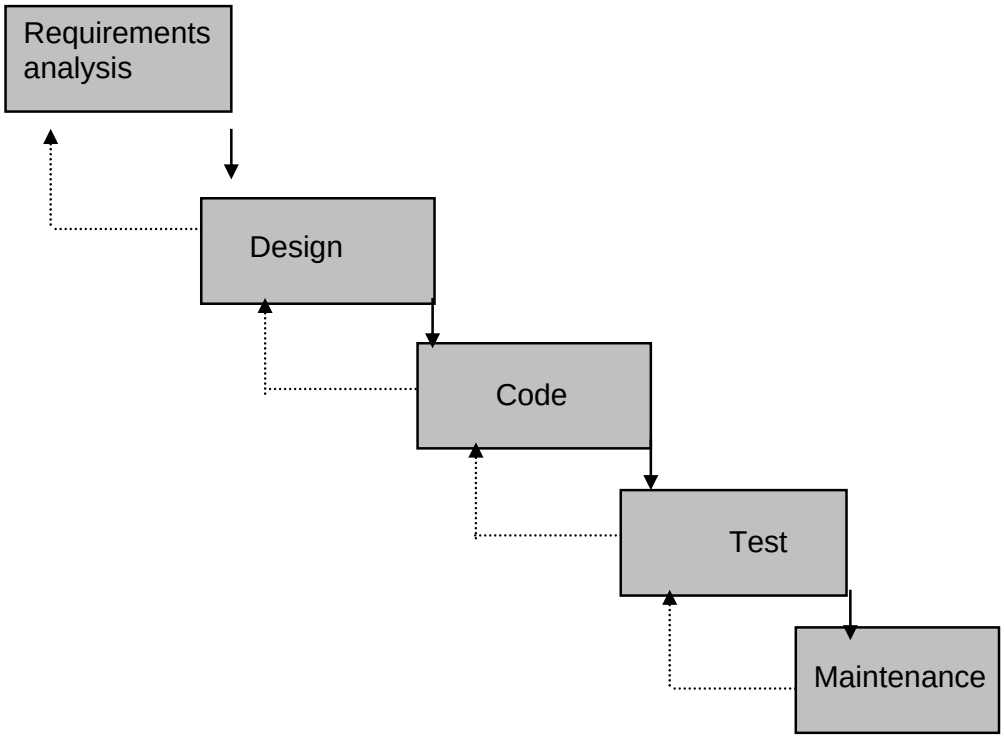
© Copyright Virtual University of Pakistan

requirements analysis has to be completed before design begin. The name given to theses steps varies, as does the precise definition of each one, but basically, the lifecycle starts with some requirements analysis, moves into design, then coding, then implementation, testing, and finally maintenance.

Flaws of waterfall model

One of the main flaws with this approach is that requirements change over time, as businesses and the environment in which they operate change rapidly. This means that it does not make sense to freeze requirements for months, or maybe years. While design and implementation are completed.

Some feedback to earlier stages was acknowledged as desirable and indeed practical soon after this lifecycle became widely used as shown in figure below. But the idea of iteration was not embedded in the waterfall's philosophy, and review sessions among developers are commonplace. However, the opportunity to review and evaluation with users was not built into this model.



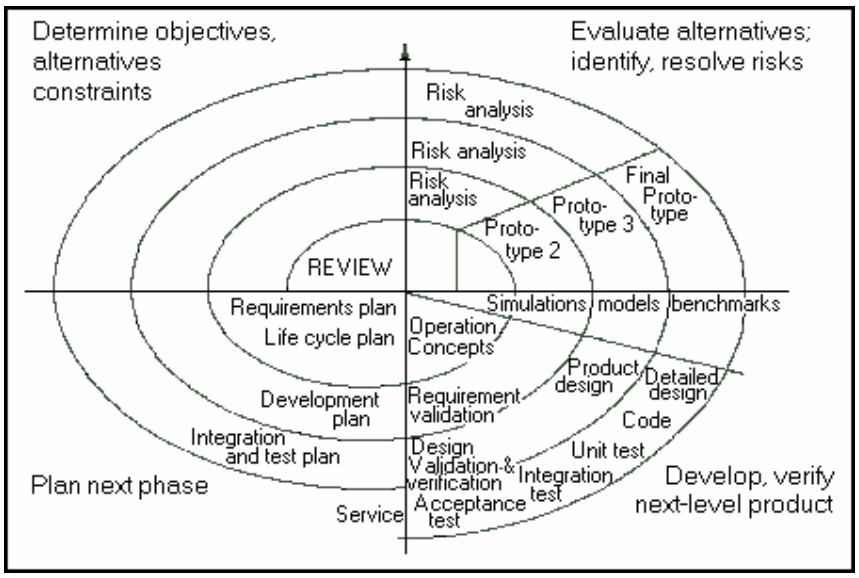
The spiral lifecycle model

For many years, the waterfall formed the basis of most software developments, but in 1988 Barry Boehm suggested the spiral model of software development. Two features of the spiral model are immediately clear from figure: risk analysis and prototyping.

The spiral model incorporates them in an iterative framework that allows ideas and progress to be repeatedly checked and evaluated. Each iteration around the spiral may be based on a different lifecycle model and may have different activities.

© Copyright Virtual University of Pakistan

In the spiral's case, it was not the need for user involvement that insured the introduction of iteration but the need to identify and control risks. In Boehm's approach, development plans and specifications that are focused on the risks involved in developing the system drive development rather than the intended functionality, as was the case with the waterfall. Unlike the waterfall, the spiral explicitly encourages alternatives to be considered, and steps in which problems or potential problems are encountered to be re-addressed.



The spiral idea has been used by others for interactive devices. A more recent version of the spiral, called the Win Win spiral model, explicitly incorporates the identification of key stakeholders and their respective "win" conditions, i.e., what will be regarded as a satisfactory outcome for each stakeholder group. A period of stakeholder negotiation to ensure a "win-win" result is included.

Rapid Application Development (RAD)

During the 1990s the drive to focus upon users became stronger and resulted in a number of new approaches to development. The Rapid Application Development (RAD) approach attempts to take a user-centered view and to minimize the risk caused by requirements changing during the course of the project. The ideas behind RAD began to emerge in the early 1990s, also in response to the inappropriate nature of the linear lifecycle models based on the waterfall. Two key features of a RAD project are:

- Time-limited cycles of approximately six months, at the end of which a system or partial system must be delivered. This is called time-boxing. In effect, this breaks down a large project into many smaller projects that can deliver product incrementally, and enhance flexibility in terms of the development techniques used and the maintainability of the final system.
- JAD (Joint Application Development) workshops in which users and developers come together to thrash out the requirements of the system. These are intensive requirements-gathering sessions in which difficult issues are faced and decisions

© Copyright Virtual University of Pakistan

are made. Representatives from each identified stakeholder group should be involved in each workshop so that all the relevant views can be heard.

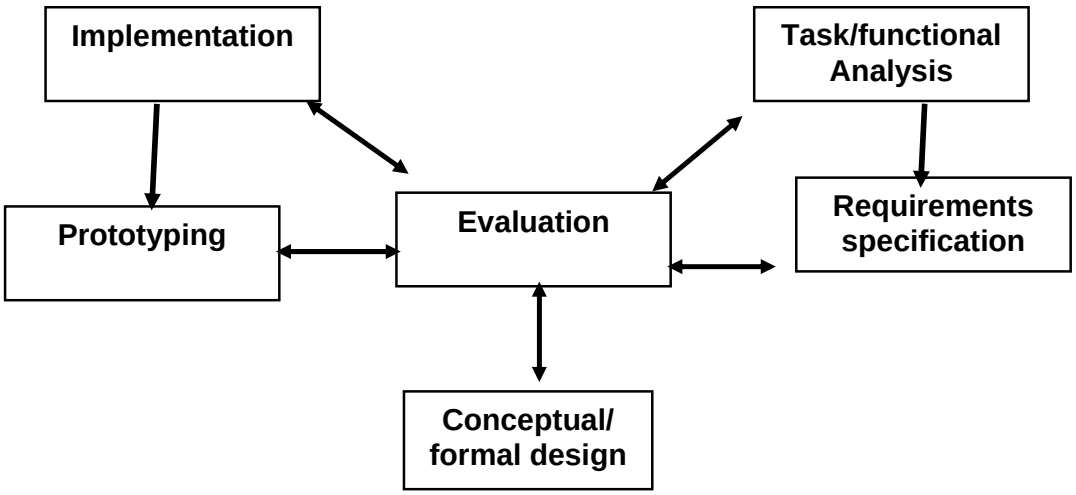
A basic RAD lifecycle has five phases (as shown in figure): project set-up, JAD workshops, iterative design and build, engineer and test final prototype, implementation review. The popularity of RAD has led to the emergence of an industry-standard RAD-based method called DSDM (Dynamic Systems Development Method). This was developed by a non-profit-making DSDM consortium made up of a group of companies that recognized the need for some standardization in the field. The first of nine principles stated as underlying DSDM is that "active user involvement is imperative." The DSDM lifecycle is more complicated than the one which shown here. It involves five phases: feasibility study, business study, functional model iteration, design and build iteration, and implementation. This is only a generic process and must be tailored for a particular organization.

17.2 Lifecycle models in HCI

Another of the traditions from which interaction design has emerged is the field of HCI. Fewer lifecycle models have arisen from this field than from software engineering and, as you would expect, they have a stronger tradition of user focus. We will discuss two of them in our lecture. The first one, the Star, was derived from empirical work on understanding how designers tackled HCI design problems. This represents a very flexible process with evaluation at its core. In contrast, the second one, the usability engineering lifecycle, shows a more structured approach and hails from the usability engineering tradition.

The Star Lifecycle model

About the same time that those involved in software engineering were looking for alternatives to the waterfall lifecycle, so too were people involved in HCI looking for alternative ways to support the design of interfaces. In 1989, the Star lifecycle model was proposed by Hartson and Hix as shown in figure.



© Copyright Virtual University of Pakistan

This emerged from some empirical work they did looking at how interface designers went about their work. They identified two different modes of activity: analytic mode and synthetic mode. The former is characterized by such notions as top-down, organizing, judicial, and formal, working from the systems view towards the user's view; the latter is characterized by such notions as bottom-up, free-thinking, creative and adhoc, working from the user's view towards the systems view. Interface designers served in software designers.

Unlike the lifecycle models introduced above, the Star lifecycle does not specify any ordering of activities. In fact, the activities are highly interconnected: you can move from any activity to any other, provided you first go through the evaluation activity. This reflects the findings of the empirical studies. Evaluation is central to this model, and whenever an activity is completed, its result(s) must be evaluated. So a project may start with requirements gathering, or it may start with evaluating an existing situation, or by analyzing existing tasks, and so on.

The Usability engineering lifecycle

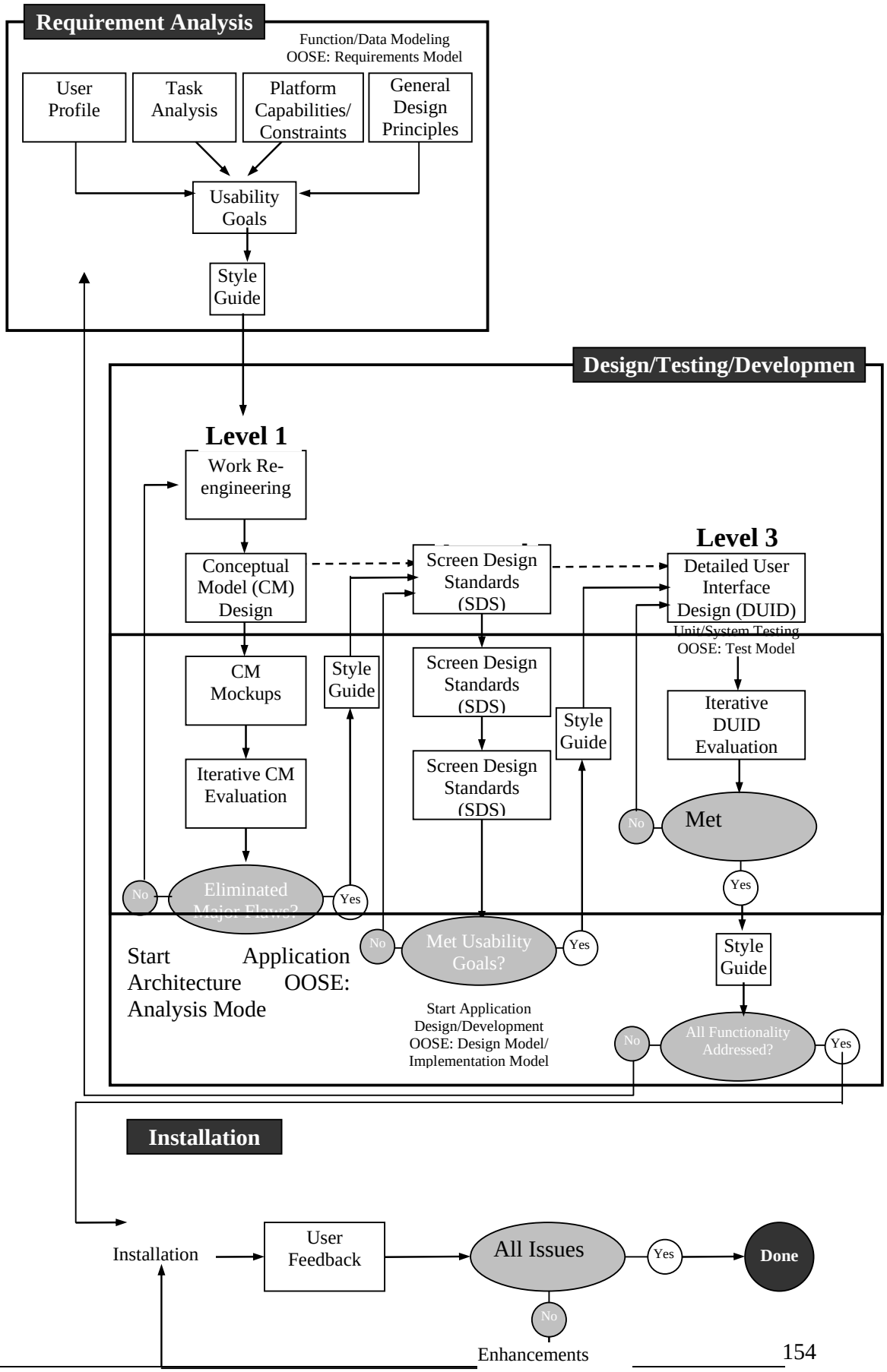
The Usability Engineering lifecycle was proposed by Deborah Mayhew in 1999. Many people have written about usability engineering, and as Mayhew herself says, "I did not invent the concept of a Usability Engineering Lifecycle. Nor did I invent any of the Usability Engineering tasks included in lifecycle….".

The Star Lifecycle Model is a flexible interface design process where activities can start in any order, but every step must include evaluation.

The Star Lifecycle Model involves the following key activities:

Task/Functional Analysis – Understanding what users need to do.
Requirements Specification – Defining what the system must do.
Conceptual/Formal Design – Creating initial design ideas and structures.
Prototyping – Building sample versions of the interface.
Implementation – Developing the actual system.
Evaluation (central step) – Testing and reviewing each activity to ensure it meets user needs.

© Copyright Virtual University of Pakistan



© Copyright Virtual University of Pakistan

However, what her lifecycle does provide is a holistic view of usability engineering and a detailed description of how to perform usability tasks, and it specifies how usability tasks can be integrated into traditional software development lifecycle. It is therefore particularly helpful for those with little or no expertise in usability to see how the tasks may be performed alongside more traditional software engineering activities. For example, Mayhew has linked the stages with an oriented software engineering that have arisen from software engineering. The lifecycle itself has essentially three tasks: requirements analysis, design/testing/development, and installation, with the middle stage being the largest and involving many subtasks as shown in figure. Note the production of a set of usability goals in the first task. Mayhew suggests that these goals be captured in a style guide that is then used throughout the project to help ensure that the usability goals are adhered to.

This lifecycle follows a similar thread to our interaction design models but includes considerably more detail. It includes stages of identifying requirements, designing, evaluating, and building prototypes. It also explicitly includes the style guide as a mechanism for capturing and disseminating the usability goals of the project. Recognizing that some projects will not require the level of structure presented in the full lifecycle, Mayhew suggests that some sub steps can be skipped if they are unnecessarily complex for the system being developed.

The Goal-Directed Design Process is a user-centered approach that focuses on understanding user goals and turning research directly into practical design solutions.

What it involves:

User research – Understand real user needs and behaviors (not just market data).

Goal definition – Identify what users want to achieve.

Modeling users – Create personas to represent user types.

Requirements definition – Translate research into clear product requirements.

Framework and interface design – Build interface structures based on user goals.

Validation – Test and refine the design to match real user needs.

The Goal-Directed Design Process

Most technology-focused companies don't have an adequate process or user-centered design, if they have a process at all. But even the more enlightened organizations, ones that can boast of an established process, come up against some critical issues that result from traditional ways of approaching the problems of research and design.

In recent years, the business community has come to recognize that user research is necessary to create good products, but the proper nature of that research is still in question in many organizations. Quantitative market research and market segmentation is quite useful for selling products, but falls short of providing critical information about how people actually use products—especially products with complex behaviors. A second problem occurs after the results have been analyzed: most traditional methods don't provide a means of translating research results into design solutions. A hundred pages of user survey data don't easily translate into a set of product requirements, and they say even less about how those requirements should be expressed in terms of a logical and appropriate interface structure. Design remains a black box: "a miracle happens here." This gap between research results and the ultimate design solution is the result of a process that doesn't connect the dots from user to final product.

Bridging the gap

As it is already discussed that the role of design in the development process needs to be change. We need to start thinking about design in new ways, and start thinking differently about how product decisions are made.

Design as product definition

Design has, unfortunately, become a limiting term in the technology industry. Design, when properly deployed identifies user requirements and defines a detailed plan for the

© Copyright Virtual University of Pakistan

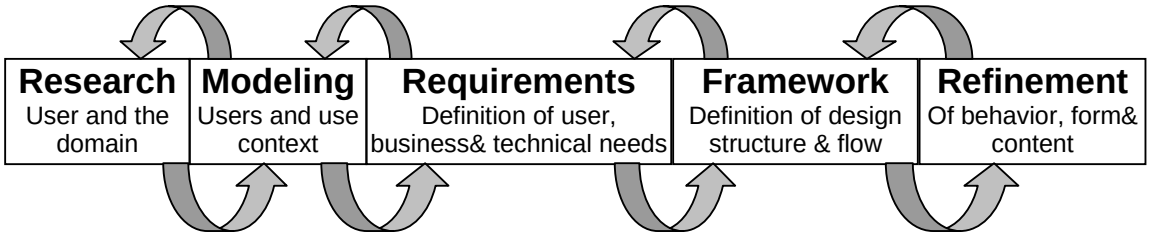
behavior and appearance of products. In other words, design provides true product definition, based on goals of users, needs of business, and constraints of technology.

Designers as researchers

If design is to become product definition, designers need to take on broader roles than that assumed in traditional design, particularly when the object of this design is complex, interactive systems. One of the problems with the current development process is that roles in the process are overspecialized: Researchers perform research, and designers perform design. The results of user and market research are analyzed by the usability and market researchers and then thrown over the transom to designers or programmers. What is missing in this model is a systematic means of translating and synthesizing the research into design solutions. One of the ways to address this problem is for designers to learn to be researchers. There is a compelling reason for involving designers in the research process. One of the most powerful tools designers bring to the table is empathy: the ability to feel what others are feeling. The direct and extensive exposure to users that proper user research entails immerses designers in he users' world, and gets them thinking about users long before they propose solutions. One of the most dangerous practices in product development is isolating designers from the users because doing so eliminates empathic knowledge. Additionally, it is often difficult for pure researchers to know what user information is really important from a design perspective. Involving designers directly in research addresses both. Although research practiced by designers takes us part of the way to Goal-Directed Design solutions, there is still a translation gap between research results and design details.

Between research and design: models, requirements, and frameworks

Few design method in common use today incorporate a means of effectively and systematically translating the knowledge gathered during research into a detailed design specification. Part of the reason for this has already been identified. Designers have historically been out of the research loop and have had to rely on third-person accounts of user behaviors and desires.



The other reason, however, is that few methods capture user behaviors in a manner that appropriately directs the definition of a product. Rather than providing information about user goals, most methods provide information at the task level. This type of information is useful for defining layout, workflow, and translation of functions into interface controls, but less useful for defining the basic framework of what a product is, what it does, and how it should meet the broad needs of the user. Instead we need explicit, systematic processes for defining user models, establishing design requirements, and translating those into a high level interaction framework.

Goal-Directed Design seeks to bridge the gap that currently exists in the digital product development process, the gap between user research and design, through a combination of new techniques and known methods brought together in more effective ways.

© Copyright Virtual University of Pakistan

Lecture 18.

Goal-Directed Design Methodologies

Learning Goals

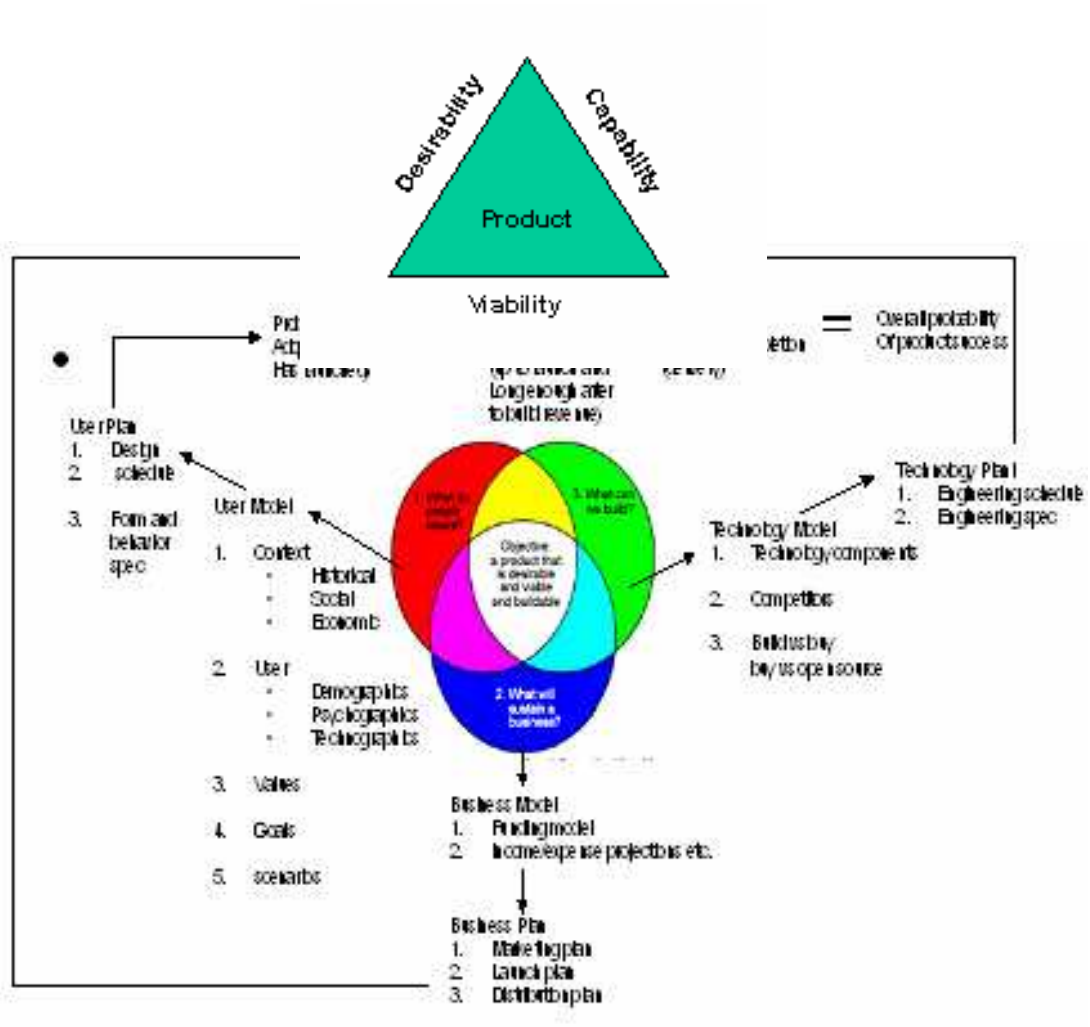
As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

Understand different phases of Goal-Directed Design Approach

18.1 Goal-Directed Design Model

Underlying the goal-directed approach to design is the premise that product must balance business and engineering concerns with user concerns.

You begin by asking, “What do people desire?” then you ask, “of the things people desire, what will sustain a business.” And finally you ask, “Of the things people desire, that will also sustain the business, what can we build?” a common trap is to focus on technology while losing the sight of viability and desirability.



© Copyright Virtual University of Pakistan

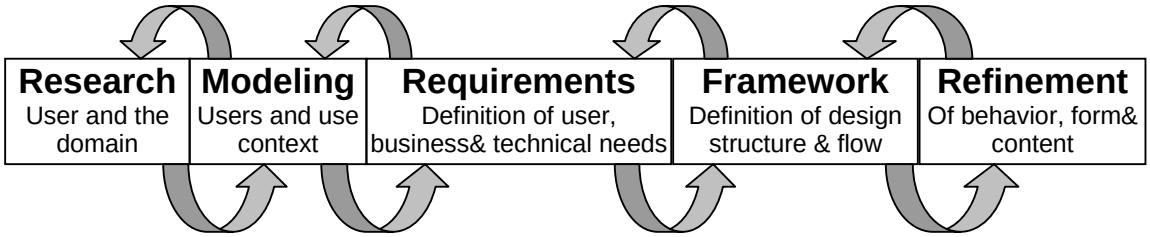
Understanding the importance of each dimension is only the beginning, which understanding must also be acted upon. We are familiar with this process along the business and technology dimension; you create a business model and then develop a business plan, and similarly create an engineering model and specification. The goal-directed design process is an analog to these planning processes. It results in a solid user model and a comprehensive interaction plan.

The user plan determines the probability that customer will adopt a product. The business plan determines the probability that business can sustain itself up to and through launch—and that sale will actually support growth thereafter. And technology plan determines the probability that the product can be made to work and actually deliverable.
Multiplying these three factors determines the overall probability that a product will be successful.

18.2 A process overview

Goal-Directed Design combines techniques of ethnography, stakeholder interviews, market research, product/literature reviews, detailed user model, scenario-based design, and a core set of interaction principles and patterns. It provides solutions that meet the needs and goals of users, while also addressing business/organizational and technical imperatives. This process can be roughly divided into five phases:

- Research
- Modeling
- Requirements Definition
- Framework Definition
- Refinement



These phases follow the five component activities of interaction design identified by Gillian Crampton Smith and Philip Tabor—understanding, abstracting, structuring, representing, and detailing—with a greater emphasis on modeling user behaviors and defining system behaviors.

Research

The research phase employs ethnographic field study techniques (observation and contextual interviews) to provide qualitative data about potential and/or actual users of the product. It also includes competitive product audits, reviews of market research and technology white papers, as well as one-on-one interviews with stakeholders, developers, subject matter experts (SMEs), and technology experts as suits the particular domain.
One of the principles out comes of field observation and user interviews are an emergent set of usage patterns—identifiable behaviors that help categorize modes of use of a

© Copyright Virtual University of Pakistan

potential or existing product. These patterns suggest goals and motivations (specific and general desired outcomes of using the product). In business and technical domains, these behavior patterns tend to map to professional roles; for consumer product, they tend to correspond to lifestyle choices. Usage patterns and the goals associated with them drive the creation of personas in the modeling phase. Market search helps select and filter for valid persons that fit corporate business models. Stakeholder interviews, literature reviews, and product audits deepen the designers' understanding of the domain and elucidate business goals and technical constraints that the design must support.

Modeling

During the modeling phase, usage and workflow patterns discovered through analysis of the field research and interviews are synthesized into domain and user models. Domain models can include information flow and workflow diagrams. User models, or personas, are detailed composite user archetypes that represent distinct groupings of behavior patterns, goals, and motivations observed and identified during the research phase.

Personas serves as the main characters in a narrative scenario-based approach to design that iterative generates design concepts in the framework definition phase, provides feedback that enforces design coherence and appropriateness in the refinement phase, and represents a powerful communication tool that helps developers and managers to understand design rationale and to prioritize features based on user needs. In he modeling phase, designers employ a variety of methodological tools to synthesize, differentiate, and prioritize personas, exploring different yupes of goals and mapping personas across ranges of behavior to ensure there are no gaps or duplications.

Specific design targets are chosen from the cast of personas through a process of comparing goals and assigning a hierarchy of priority based on how broadly each persona's goals encompass the goals of other personas. A process of designating persona types determines the amount of influence each persona has on the eventual form and behavior of the design.

Possible user persona type designations include:

- Primary: the persona's needs are sufficiently unique to require a distinct interface form and behavior
- Secondary: primary interface serves the needs of the persona with a minor modification or addition
- Supplement: the persona's needs are fully satisfied by a primary interface
- Served: the persona is not an actual user of the product, but is indirectly affected by it and its use
- Negative: the persona is created as an explicit, rhetorical example of whom not to design for

Requirements definition

Design methods employed by teams during the requirements definition phase provides the much-needed connection between user and other models and the framework of the design. This phase employs scenario-based design methods, with the important innovation of focusing the scenarios not on user tasks in the abstract, but first and foremost of meeting the goals and needs of specific user personas. Personas provide and

© Copyright Virtual University of Pakistan

understanding of which tasks are truly important and why, leading to an interface that minimize necessary tasks while maximizing return. Personas become the main characters of these scenarios, and the designers explore the design space via a form of role-playing.
For each interface/primary persona the process of design in the requirements definition phase involves an analysis of persona data and functional needs, prioritized and informed by persona goals, behaviors, and interactions with other personas in various contexts.

This analysis is accomplished through an iteratively refined context scenario that start with a "day in the life" of the persona using the product, describing high-level product touch points, and thereafter successively defining detail at ever-deepening levels. As this iteration occurs, both business goals and technical constraints are also considered and balanced with personas goals and needs. The output of this process is a requirements definition that balances user, business, and technical requirements of the design to follow.

Framework definition

In the framework definition phase, teams synthesize an interaction framework by employing two other critical methodological tools in conjunction with context scenarios. The first is a set of general interaction design principles that, like their visual design counterparts, provide guidance in determining appropriate system behavior in a variety of contexts

The second critical methodological tool is a set of interaction design patterns that encode general solutions (with variations dependent on context) to classes of previously analyzed problems. These patterns bear close resemblance to the concept of architectural design patterns developed by Christopher Alexander. Interaction design patterns are hierarchically organized and continuously evolve as new contexts arise. Rather than stifling designer creativity, they often provide needed leverage to approach difficult problems with proven design knowledge.

After data and functional needs are described at this high level, they are translated into design elements according to interaction principles and then organized using patterns and principles into design sketches and behavior descriptions. The output of this process is an interaction framework definition, a stable design concept that provides the logical and gross formal structure for the detail to come. Successive iterations of more narrowly focused scenarios provide this detail in ht refinement phase. The approach is often a balance of top-down design and bottom-up design.

Refinement

The refinement phase proceeds similarly to the framework definition phase, but with greater focus on task coherence, using key path and validation scenarios focused on storyboarding paths through the interface in high detail. The culmination of the refinement phase is the detailed documentation of the design, a form and behavior specification, delivered in either paper or interactive media as context dictates.

Goals, not features, are key to product success

Programmers and engineers—people who are intrigued by technology—share a strong tendency to think about products in terms of functions and features. This is only natural, as this is how developers build software: function-by-function. The problem is that this is not how users want to use it.

© Copyright Virtual University of Pakistan

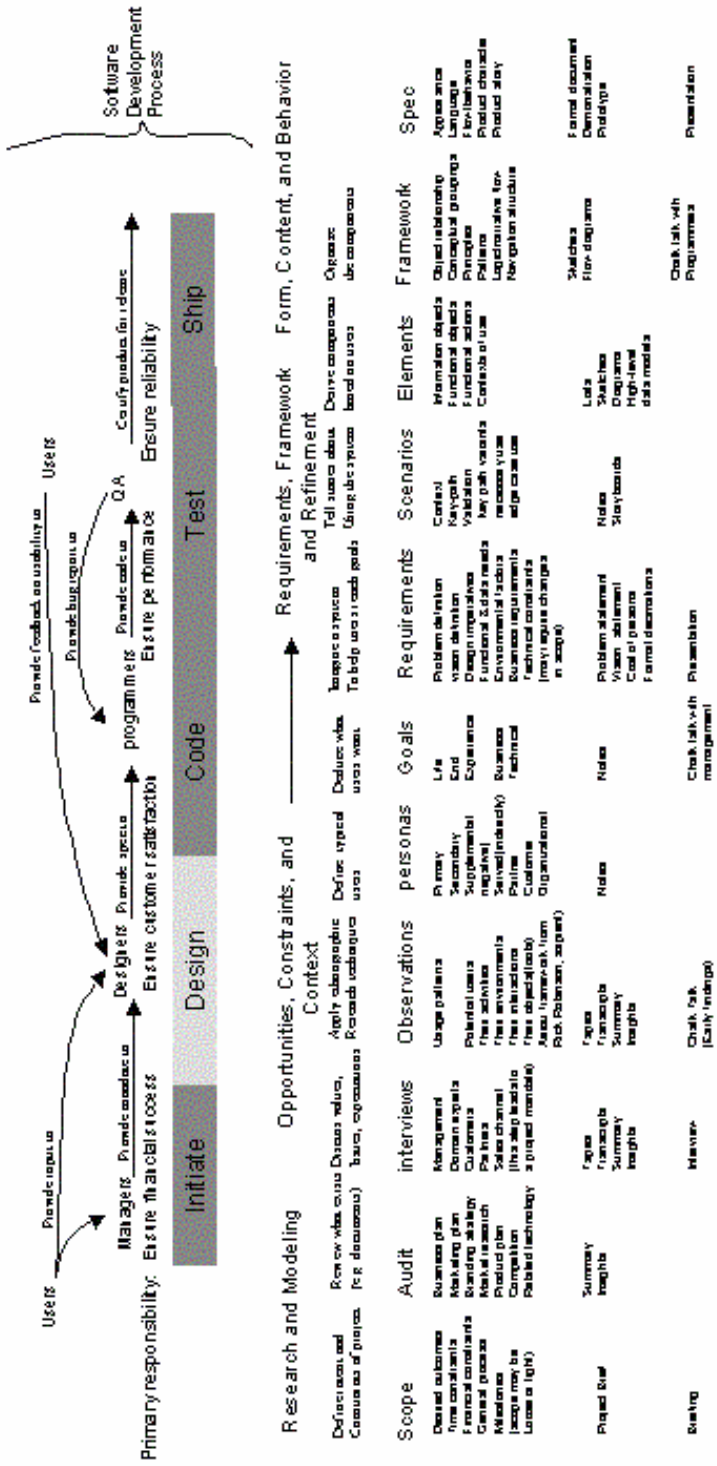
The decision about whether a feature should be included in a product shouldn't rest on its technological understandings. The driving force behind the decision should never be simply that we have the technical capability to do it. The deciding factor should be whether that feature directly, or indirectly, helps to achieve the goals of the user while still meeting the needs of the business.

The successful interaction designer must be sensitive to user's goals amid the pressures and chaos of the product-development cycle. The Goal-Directed process, with its clear rationale for design decisions, makes persuading engineers easier, keeps marketing and management stakeholders in the loop, and ensures that the design in question isn't just guesswork, or a reflection of the team members' personal preferences.

Most computer users know all too well that opening the shrink wrap on a new software product augurs several days of frustration and disappointment spent learning the new interface. On the other hand, many experienced users of a program may find themselves continually frustrated because the program always treats them like rank beginners. It seems impossible to find the right balance between catering to the needs of the first-timer and the needs of the expert.

One of the eternal conundrums of interaction and interface design is deciding how to address the needs of both beginning users and expert users with a single interface. Some programmers and designers choose to abandon this idea completely, choosing instead to create software with a beginner mode and an expert mode, the former usually being an oversimplified and underpowered subset of the latter. Of course, nobody wants to be caught dead using software in beginner mode, but the leap from there to expert mode is usually off a rather tall cliff into a shark-infested moat of implementation-model design. What, then, is the answer? The solution to this predicament lies in a different understanding of the way user's master new concepts and tasks.

© Copyright Virtual University of Pakistan



18.3 Types of users

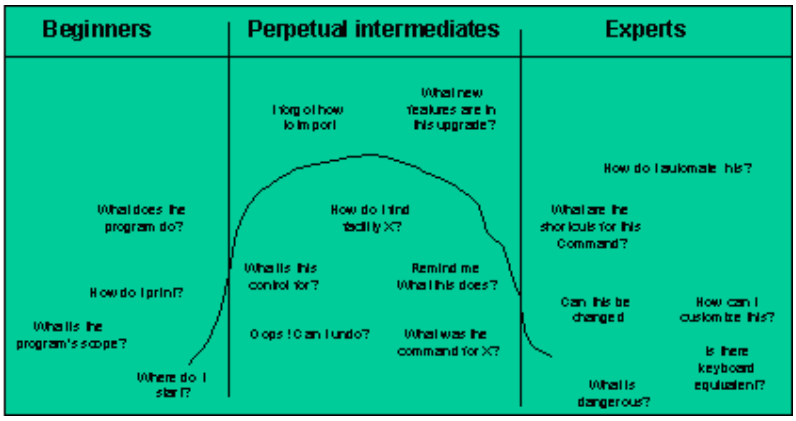
Most users are neither beginners nor experts; instead they are intermediates.
The experience level of people performing an activity requiring knowledge or skill, if we graph number of people against skill level, a relatively small number of beginners are on

© Copyright Virtual University of Pakistan

the left side, a few experts are on the right, and the majority—intermediate users—are in the center.

Statistics don't tell the whole story, however, the bell curve is a snapshot in time, and although most intermediate tend to stay in that category, the beginners do not remain beginners for very long. The difficulty of maintaining a high level of expertise also means that experts come and go rapidly, but beginners change even more rabidly. Both beginners and experts tend over time to gravitate towards intermediacy.

Although everybody spends some minimum time as a beginner, nobody remains in that state for long. People don't like to be incompetent; and beginners, by definition, are incompetent. Conversely, learning and improving is rewarding, so beginners become intermediates very quickly—or they drop out altogether. All skiers, for example, send time as beginners, but those who find they don't rapidly progress beyond more-falling-than-skiing quickly abandon the sport. The rest soon move off of the bunny slopes onto the regular runs. Only a few ever make it onto the double-black diamond runs for experts.



The occupants of the beginner end of the curve will either migrate into the center bulge of intermediates, or they will drop off the graph altogether and find some product or activity in which they can migrate into intermediacy. Most users thus remain in a perpetual state of adequacy striving for fluency, with their skills ebbing and flowing like the tides depending on how frequently they use the program. Larry Constantine first identified the importance of designing for intermediates, and in his book Software for Use; he refers to such users as improving intermediates. The term perpetual intermediates are preferred, because although beginners quickly improve to become intermediates, they seldom go on to become experts.

A good ski resort bas a gentle slope for learning and a few expert runs to really challenge the serious skiers. But if the resort wants to stay in business, it will cater to the perpetual intermediate skier, without scaring off the beginner or insulting the expert. The beginner must find it easy to matriculate into the world of intermediacy, and the expert must not find his vertical runs obstructed by aids for bewildered perpetual intermediates.

A well-balanced user interface takes the same approach. It doesn't cater to the beginner or to the experts, but rather devotes the bulk of its efforts to satisfying the perpetual intermediate. At the same time, it avoids offending either of its smaller constituencies, recognizing that they are both vital.

© Copyright Virtual University of Pakistan

Most users in this middle state would like to learn more about the program but usually don't have the time. Occasionally, the opportunity to do so will surface. Sometimes these intermediates use the product extensively for weeks at a time to complete a big project. During this time, they learn new things about the program. Their knowledge grows beyond its previous boundaries.

Sometimes, however, they do not use the program for months at a time and forget significant portions of what they knew. When they return to the program, they are not beginners, but they will need reminders to jog their memory back to its former state.
If a user finds himself not satisfactorily progressing beyond the beginner stage after only a few hours, he will often abandon the program altogether and find another to take its place. No one is willing to remain incompetent at a task for long.

Optimizing for intermediates

Now let's contrast our bell curve of intermediates with the way that software is developed. Programmers qualify as experts in the software they code because they have to explore every possible use case, no matter how obscure and unlikely, to create program code to handle it. Their natural tendency is to design implementation model software with every possible option given equal emphasis in the interaction, which they, as experts, have no problem understanding.

All the same, time, sales, marketing, and management—none of whom are likely to be expert users or even intermediates—demonstrate the product to customers, reporters, partners, and investors who are themselves unfamiliar with the product. Because of their constant exposure to beginners, these professionals have a strongly biased view of the user community. Therefore, it comes as no surprise that sales and marketing folks lobby for bending the interface to serve beginners. They demand that training wheels be attached to the product to help out the struggling beginner.

Programmers create interaction suitable only for experts, while the marketers demand interactions suitable only for beginners, but the largest, most stable, and most important group of users is the intermediate group.

It's amazing to think that the majority of real users are typically ignored, but more often than not that is the case. You can see it in many enterprise and commercial software-based products. The overall design biases them towards expert users, while at the same time, cumbersome tools like wizards and Clippy are rafted on the meet the marketing department's perception of new users. Experts rarely use them, and beginners soon desire these embarrassing reminders of their ignorance. But the perpetual intermediate majority is perpetually stuck with them.

Our goal should be neither to pander to beginners nor to rush intermediates into expertise. Our goal is threefold: to rapidly and painlessly get beginners into intermediacy; to avoid putting obstacles in the way of those intermediates who want to become experts; an most of all, to keep perpetual intermediates happy as they stay firmly in the middle of the skill spectrum.

© Copyright Virtual University of Pakistan

It is required to spend more time making our programs powerful and easy to use for perpetual intermediate users. Beginners and experts are also accommodated but not to the discomfort of the largest segment of users.

What perpetual intermediates need

Perpetual intermediates need access to tools. They don't need scope and purpose explained to them because they already know these things. Tooltips are the perfect perpetual intermediate idiom. Tooltips say nothing about scope and purpose and meaning; they only state function in the briefest of idioms, consuming the least amount of video space in the process.

Perpetual intermediates know how to use reference materials. They are motivated to dig deeper and learn, as long as they don't have to tackle too much at once. This means that online help is a perpetual intermediate tool. They use it by way of the index, so that part of help must be very comprehensive.

Perpetual intermediates will be establishing the functions that they use with regularity and those that they only use rarely. The user may experiment with obscure features, but he will soon identify—probability subconsciously—his frequently used working set. The user will demand that the tools in his working set are placed front-and-center in the user interface, easy to find and to remember.

Perpetual intermediates usually know that advanced features exist, even though they may not need them or know how to use them. But the knowledge that they are there is reassuring to the perpetual intermediate, convincing him that he made the right choice investing in this program. The average skier may find it reassuring to know that there is a really scary black diamond expert runs just beyond those trees, even if she never intends to use it. It gives her something to aspire to and dream about.

You program's code must provide for both rank amateurs and all the possible cases an expert might encounter. Don't let this technical requirement influence your design thinking. Yes, you must apply the bulk of your talents, time, and resources to designing the best interaction possible for your most representative users: the perpetual intermediates.

© Copyright Virtual University of Pakistan

Lecture 19.

User Research Part-I

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the difference in qualitative and quantitative research
- Discuss in detail qualitative research technique

The outcome of any design effort must ultimately be judged by how successfully it meets the requirements of both the user and the organization that commissioned it. No matter how skillful and creative the designer. If she does not have a clear and detailed knowledge of the users she is designing for, what the constraints of the problem are, and what business or organizational goals the design is hoping to achieve, she will have little chance of success.

What and how questions like these are best answered by qualitative research, not metrics or demographics (though these also have their purpose). There are many types of qualitative research, each of which plays an important role in filling in a picture of the design landscape of a product.

Qualitative versus Quantitative Research

Research is a word that most people associate with science and objectivity. This association isn't incorrect, but it biases many people towards the notion that the only valid sort of research is the kind that yields the supposed ultimate in objectivity: quantitative data. The notion that numbers don't lie is prevalent in the business and engineering communities, even though we all rationally understand that numbers—especially numbers ascribed to human activities—can be manipulated or reinterpreted at least as dramatically as words.

Data gathered by the hard sciences like physics is simply different from that gathered on human activities: electrons don't have moods that vary from minute to minute, and the tight controls physicists place on their experiments to isolate observed behaviors are next to impossible in the social sciences. Any attempt to reduce human behavior to statistics is likely to overlook important nuances, which though they might not directly affect business plans, do make an enormous difference to the design of products. Quantitative research can only answer questions about how much or how many along a few reductive axes. Qualitative research can tell you about what, how and why in rich, multivariate detail.

Social scientists have long realized that human behaviors are too complex and subject to too many variables to rely solely on quantitative data to understand them. Usability practitioners, borrowing techniques from anthropology and other social sciences, have

© Copyright Virtual University of Pakistan

developed many alternative methods for gathering useful data on user behaviors to a more pragmatic end: to help create products that better serve user need.

The value of qualitative research

Qualitative research helps us understand the domain, context and constraints of a product in different, more useful ways than quantitative research do. It also quickly helps us identify patterns of behavior among users and potential users of a product much more quickly and easily than would be possible with quantitative approaches. In particular, qualitative research helps us understand:

- Existing products and how they are used.
- Potential users of new or existing products, and how they currently approach activities and problems the new product design hopes to address
- Technical, business, and environmental contexts—the domain—of the product to be designed
- Vocabulary and other social aspects of the domain in question

Qualitative research cab also helps the progress of design projects by:

- Providing credibility and authority to the design team, because design decisions can be traced to research results
- Uniting the team with a common understanding of domain issues and user concerns
- Empowering management to make more informed decisions about product design issues that would otherwise be based on guesswork or personal preference

It is the experienced that qualitative method, in addition to the benefits described above, tend to be faster, less expensive, more flexible, and more likely than their quantitative counterparts t provide useful answers to impotent questions that leads t superior design:

- What problems are people encountering with their current ways of doing what the product hopes to do?
- Into what broader contexts in people's lives does the product fit and how?
- What are the basic goals people have in using the product, and what basic tasks help people accomplish them?

19.1 Types of qualitative research

Social science and usability texts are full of methods and techniques for conducting qualitative research. We will discuss following qualitative research techniques:

- Stakeholder interviews
- Subject matter expert (SME) interviews
- User and customer interviews
- User observation/ethnographic field studies
- Literature review
- Product/prototype and competitive audits

© Copyright Virtual University of Pakistan

Stakeholder interviews

Research for any new product design, through it must end with understanding the user, should start by understanding the business and technical context in which the product will be built. This is necessary not only to ensure a viable and feasible end result, but also to provide a common language and understanding among the design team, management, and engineering teams.

Stakeholders are any key members of the organization commissioning the design work, and typically include managers and key contributors from engineering, sales, product marketing, marketing communications, customer support, and usability. They may also include similar people from other organizations in business partnership with the commissioning organization, and executives. Interviews with stakeholders should occur before any user research begins.

It is usually most effective to interview each stakeholder one-on-one, rather than in a larger, cross-departmental group. A one-on-one setting promotes candor on the part of the stakeholder, and ensure that individual views are not lost in a crowd. Interviews need not last longer than about an hour, though follow-up meetings may be called for if a particular stakeholder is identified as an exceptionally valuable source of information.

The type of information that is important to gather from stakeholders includes:

- What is the preliminary vision of the product from each stakeholder perspective? As in the fable of the blind men and the elephant, you may find that each business department has a slightly different and slightly incomplete perspective on the product to be designed. Part of the design approach must therefore involve harmonizing these perspectives with those of users and customers.
- What is the budget and schedule? The answer to this question often provides a reality check on the scope of the design effort and provides a decision point for management if user research indicates a greater scope is required.
- What are the technical constraints? Another important determinant of design scope is a firm understanding of what is technically feasible given budget, time, and technology.
- What are the business drivers? It is important for the design team to understand what the business is trying to accomplish. This again leads to a decision point, should user research indicate a conflict between business and user needs. The design must, as much as possible, create a win-win situation for users, customers, and providers of the product.
- What are the stakeholders' perceptions of the user? Stakeholders who have relationships with users (such as customer support representative) may have important insights on users that will help you to formulate your user research plan. You may also find that there are significant disconnects between some stakeholders' perceptions of their users and what you discover in your research. This information can become an important decision point for management later in the process.

Understanding these issues and their impact on design solutions helps you as a designer to better serve your customer, as well as users of the product. Building consensus internally will help you to articulate issues that the business as a whole may not identified, build

© Copyright Virtual University of Pakistan

internal consensus that is critical for decision making later in the design process, and build credibility for your design team.

Subject matter expert (SME) interviews

Some stakeholders may also be subject matter experts (SMEs): experts on the domain within which the product you are designing will operate. Most SMEs were users of the product or its predecessors at one time, and may now be trainers, managers, or consultants. Often they are experts hired by stakeholders, rather than stakeholders themselves. Similar to stakeholders, SMEs can provide valuable perspective on a product and its users, but designers should be careful to recognize that SMEs represent a somewhat skewed perspective. Some points to consider about using SMEs are:

- SMEs are expert users. Their long experience with a product or its domain mean that they may have grown accustomed to current interactions. They may also lean towards expert controls rather than interactions designed for perpetual intermediate perspective. SMEs are often not current users of the product, and may have more of a management perspective.
- SMEs are knowledgeable, but they aren't designers. They may have many ideas on how to improve a product. Some of these may be valid and valuable, but the most useful pieces of information to glean from these suggestions are the causative problems that lead to their proposed solutions.
- SMEs are necessary in complex or specialized domains such as medical, scientific, or financial services. If you are designing for a technical or otherwise specialized domain, you will likely need some guidance from SMEs, unless you are one yourself. Use SMEs to get information on complex regulations and industry best practices. SME knowledge of user roles and characteristics is critical for planning user research in complex domains.
- You will want access to SMEs throughout the design process. If your product domain requires use of SMEs, you should be able to bring them in at different stages of the design to help perform reality checks on design details. Make sure that you secure this access in your early interviews.

User and customer interviews

It is easy to confuse users with customers. For consumer products, customers are often the same as users, but in corporate or technical domain, users and customers rarely describe the same sets of people. Although both groups should be interviewed, each has its own perspective on the product that needs to be factored quite differently into an eventual design.

Customers of a product are those people who make the decision to purchase it. For consumer product, customers are frequently users of the product; although for products aimed at children or teens, the customers are parents or other adult supervisors of children. In the case of most enterprise or technical products, the customer is someone very different from the user—often an IT manager—with distinct goals and needs. It's important to understand customers and their goals in order to make a product viable. It is also important to realize that customers seldom actually use the product themselves, and when they do, they use it quite differently than the way their users do.

© Copyright Virtual University of Pakistan

When interviewing customers, you will want to understand:

- Their goals in purchasing the product
- Their frustrations with current solutions
- Their decision process for purchasing a product of the type you're designing
- Their role in installation, maintenance, and management of the product
- Domain related issues and vocabulary

Like SMEs, customers may have many opinions about how to improve the design of the product. It is important to analyze these suggestions, as in the case of SMEs, to determine what issues or problems underline the ideas offered, because better, more integrated solutions become evident later in the design process.

Users of a product should be the main focus of the design effort. They are the people (not their managers or support team) who are personally trying to accomplish something with the product. Potential users are people who do not currently use the product, but who are good candidates for using it in the future. A good set of user interviews includes both current users (if the product already exists and is being revised) and potential users (users of competitive products and non-automated systems of appropriate). Information we are interested in learning from users includes:

- Problems and frustrations with the product (or analogous system if they are potential users)
- The context of how the product fits into their lives or workflow: when, why, and how the product is used, that is, patterns of user behavior with the product.
- Domain knowledge from a user perspective: what do users need to know to accomplish their jobs
- A basic understanding of the users' current tasks: both those the product requires and those it doesn't support
- A clear understanding of user goals: their motivations and expectations concerning use of the product

User observation

Most people are incapable of accurately assessing their own behaviors, especially outside of the context of their activities. It then follows that interviews performed outside the context of the situations the designer hopes to document will yield less complete and less accurate data. Basically, you can talk to users about how they think they behave, or you can observe it first hand. The latter route provides superior results.

Many usability professionals make use of technological aids such as audio or video recorders to capture what users say and do. Care must be taken not to make these technologies too obtrusive: otherwise the users will be distracted and behave differently than they would off-tape.

Perhaps the most effective technique for gathering qualitative user data combines interviews and observation, allowing the designer to ask clarifying questions and direct inquiries about situations and behaviors they observe in real-time.

© Copyright Virtual University of Pakistan

Literature review

In parallel with stakeholder interviews, the design team should review any literature pertaining to the product or its domain. This can and should include product marketing plans, market research, technology specifications and white papers, business and technical journal articles in the domain, competitive studies. Web searches for related and competing products and news, usability study results and metrics, and customer support data such as call center statistics.

The design team should collect this literature, use it as a basis for developing questions to ask stakeholders and SMEs, and later use it to supply addition domain knowledge and vocabulary, and to check against compiled user data.

Product and competitive audits

Also in parallel to stakeholder and SME interviews, it is often quite helpful for the design team to examine any existing version or prototype of the product, as well as its chief competitors. Doing so gives the design team a sense of the state of the art, and provides fuel for questions during the interviews. The design team, ideally, should engage in an informal heuristic or expert review of both the current and competitive interfaces, comparing each against interaction and visual design principles. This procedure both familiarizes the team with the strengths and limitations of what is currently available to users, and provides a general idea of the current functional scope of the product.

© Copyright Virtual University of Pakistan

Lecture 20.

User Research Part-II

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the User-Centered approach
- Discuss in detail the ethnographic interviews
- Understand how to prepare for ethnographic interviews

In our last lecture we were studying the qualitative research techniques. Today we will discuss last technique, ethnographic field study. But, first let us look at the user-centered design approach.

20.1 User-Centered Approach

The user-centered approach means that the real users and their goals, not just technology, should be the driving force behind development of a product. As a consequence, a well-designed system should make the most of human skill and judgment, should be directly relevant to the work in hand, and should support rather than constrain the user. This is less technique and more a philosophy.

In 1985, Gould and Lewis laid down three principles they believed would lead to a "useful and easy to use computer system." These are very similar to the three key characteristics of interaction design.

- Early focus on users and tasks: This means first understanding who the users will be by directly studying their cognitive, behavioral, anthropomorphic, and attitudinal characteristics. This required observing users doing their normal tasks, studying the nature of those tasks, and then involving users in the design process.
- Empirical measurement: early in development, the reactions and performance of intended users to printed scenarios, manuals, etc, is observed and measured. Later on, users interact with simulations and prototypes and their performance and reactions are observed, recorded and analyzed.
- Iterative design: when problems are found in user testing, they are fixed and then more tests and observations are carried out to see the effects f the fixes. This means that design and development is iterative, with cycles of "design, test, measure, and redesign" being repeated as often as necessary.

Iteration is something, which is emphasized in user-centered design and is now widely accepted that iteration is required. When Gould and Lewis wrote their paper, however, the iterative nature of design was not accepted by most developers. In fact, they comment in their paper how "obvious" these principles are, and remark that when they started

© Copyright Virtual University of Pakistan

recommending these t designers, the designers' reactions implied that these principles were indeed obvious.

Applying ethnography in design

Ethnography is a method that comes originally from anthropology and literally means "writing the culture". It has been used in the social sciences to display the social organization of activities, and hence to understand work. It aims to find the order within an activity rather than impose any framework of interpretation on it. It is a broad-based approach in which users are observed as they go about their normal activities. The observers immerse themselves in the users' environment and participate in their day-to-day work, joining in conversations, attending meetings, reading documents, and so on. The aim of an ethnographic study is to make the implicit explicit. Those in the situation, the users in this case, are so familiar with their surroundings and their daily tasks that they often don't see the importance of familiar actions or happenings, and hence don't remark upon them in interviews or other data-gathering sessions.

There are different ways in which this method can be associated with design. Beynon-Davies has suggested that ethnography can be associated with the development as "ethnography of", "ethnography for", and "ethnography within." Ethnography of development refers to studies of developers themselves and their workplace, with the aim of understanding the practices of development (e.g. Button and Sharrock). Ethnography for development yields ethnographic studies that can be used as a resource for development, e.g., studies of organizational work. Ethnography within software development is the most common form of study; here the techniques associated with ethnography are integrated into methods and approaches for development.

Because of the very nature of the ethnography experience, it is very difficult to describe explicitly what data is collected through such an exercise. It is an experience rather than a data-collection exercise. However, the experience must be shared with other team members, and therefore needs to be documented and rationalized.

Studying the context of work and watching work being done reveals information that might be missed by other methods that concentrative on asking about work away from its natural setting. For example, it can shed light on how people do the "real" work as opposed to the formal procedures that you 'd find in documentation; the nature and purpose of collaboration, awareness of other's work, and implicit goals that may not even be recognized by the workers themselves. For example, Heath et al. has been exploring the implications of ethnographic studies of real-world setting for the design of cooperative systems. They studied medical centers, architects' practices, and TV and radio studios.

In one of their studies Heath et al. looked at how dealers in a stock exchange work together. A main motivation was to see whether proposed technological support for market trading was indeed suitable for that particular setting. One of the tasks examined in detail was the process f writing ticket to record deals. It had been commented upon earlier by others that this process of deal capture, using "old-fashioned" paper and pencil technology, was currently time-consuming and prone to error. Based on this finding, it had been further suggested that the existing way of making deals could be improved by

© Copyright Virtual University of Pakistan

introducing new technologies, including touch screens to input the details of transactions, and headphones to eliminate distracting external noise.

However, when Heath et al. began observing the deal capture in practice, they quickly discovered that these proposals were misguided. In particular, they warned that these new technologies would destroy the very means by which the traders currently communicate and keep informed of what others are up to. The touch screens would reduce the availability of information to others on how deals were progressing; while headphones would impede the dealers' ability to inadvertently monitoring of other dealers' actions was central to the way deals are done. Moreover, if any dealers failed to keep up with what the other dealers were doing by continuously monitoring them, it was likely to affect their position in the market, which ultimately could prove very costly to the bank they were working for.

Hence, the ethnographic study proved to be very useful in warning against attempts to integrate new technologies into a workplace without thinking through the implications for the work practice. As an alternative, Heath et al. suggested pen-based mobile systems with gesture recognition that could allow deals to be made efficiently while also allowing the other dealers to continue to monitor one another unobtrusively. Hughes et al state that "doing" ethnography is about being reasonable, courteous and unthreatening, and interested in what's happening. Training and practice are required to produce good ethnographies.

Collecting ethnographic data is not hard although it may seem a little bewildering to those accustomed to using a frame of reference to focus the data collection rather that letting the frame of reference arise from the available data. You collect what is available, what is "ordinary", what it is that people do, say, how they work. The data collected therefore has many forms: documents, notes of your own, pictures, room layouts.
In some way, the goals of design and the goals of ethnography are at opposite ends of a spectrum. Design is concerned with abstraction and rationalization. Ethnography, on the other hand, is about detail. An ethnographer's account will be concerned with the minutiae of observation, while a designer is looking for useful abstractions that can be used to inform design. One of the difficulties faced by those wishing to use this very powerful technique is how to harness the data gathered in a form that can be used in design.

20.2 Ethnography framework

Ethnographic framework has been developed specifically to help structure the presentation of ethnographies in a way that enables designers to user them. This framework has three dimensions

1. Distributed co-ordination
2. Plans and procedures
3. Awareness of work

1. Distributed co-ordination

The distributed co-ordination dimension focuses on the distributed nature of the tasks and activities, and the means and mechanisms by which they are coordinated. This has implications for the kind of automated support required.

© Copyright Virtual University of Pakistan

2. Plans and procedures

The plans and procedures dimension focuses on the organizational support for the work, such as workflow models and organizational charts, and how these are used to support the work. Understanding this aspect impacts on how the system is designed to utilize this kind of support.

3. Awareness of work

The awareness of work dimension focuses on how people keep themselves aware of others' work. No one works in isolation, and it has been shown that being aware of others' actions and work activities can be a crucial element of doing a good job. In the stock market example this was one aspect that ethnographers identified. Implications here relate to the sharing of information.

Rather than taking data from ethnographers and interpreting this in design, an alternative approach is to train developers to collect ethnographic data themselves. This has the advantage of giving the designers first-hand experience of the situation. Telling someone how to perform a task, or explaining what an experience is like is very difficult from showing him or her or even gaining the experience themselves. Finding people with the skills of ethnographers and interaction designers may be difficult, but it is possible to provide notational and procedural mechanisms to allow designers to gain some of the insights first-hand. Two methods described bellow give such support.

- Coherence
- Contextual design

Coherence

The coherence method combines experiences of using ethnography to inform design with developments in requirements engineering. Specifically, it is intended to integrate social analysis with object-oriented analysis from software engineering. Coherence does not prescribe how to move form the social analysis to use cases, but claims that presenting the data from a ethnographic study based around a set of "viewpoints" and "concerns" facilitated the identification of the product's most impotent use cases.

Viewpoints and concerns

Coherence builds upon the framework introduced above and provides a set of focus questions for each of the three dimensions, here called "viewpoints". The focus questions are intended to guide the observer to particular aspects of the workplace. They can be used as a starting point to which other questions may be added as experience in the domain and the method increase.

In addition to viewpoints, Coherence has a set of concerns and associated questions. Concerns are a kind of goal, and they represent criteria that guide the requirements activity. These concerns are addressed within each appropriate viewpoint. One of first tasks is to determine whether the concern is indeed relevant to the viewpoint. If it is relevant, then a set of elaboration questions is used to explore the concern further. The concerns, which have arisen from experience of using ethnography in systems design, are:

- Paper work and computer work
- Skill and the use of local knowledge

© Copyright Virtual University of Pakistan

- Spatial and temporal organization
- Organizational memory

Paperwork and computer work

These are embodiments of plans and procedures, and at the same time are a mechanism for developing and sharing an awareness of work.

Skill and the use of local knowledge

This refers to the "workarounds" that are developed in organizations and are at the heart of how the real work gets done.

Spatial and temporal organization

This concern looks at the physical layout of the workplace and areas where time is important.

Organizational memory

Formal documents are not the only way in which things are remembered within an organization. Individuals may keep their own records, or there maybe local gurus.

Contextual design

Contextual design was another technique that was developed to handle the collection and interpretation of data from fieldwork with the intention of building a software-based product. It provides a structured approach to gathering and representing information from fieldwork such as ethnography, with the purpose of feeding it into design.
Contextual design has seven parts:

- Contextual inquiry
- Work modeling, consolidation
- Work redesign
- User environment design
- Mockup
- Test with customers
- Putting it into practice

Contextual inquiry

Contextual inquiry, according to Beyer and Holtzblatt, is based on a master-apprentice model of learning: observing and asking questions of the users as if she is the master craftsman and he interviews the new apprentice. Beyer and Holtzblatt also enumerate four basic principles for engaging in ethnographic interview:

Context:

Rather than interviewing the user in a clean white room, it is important to interact with and observe the user in their normal work environment, or whatever physical context is appropriate for the product. Observing users as they perform activities and questioning them in their own environment, filled with the artifacts they use each day, can bring the all-important details of their behaviors to light.

© Copyright Virtual University of Pakistan

Partnership:

The interview and observation should take the tone of a collaborative exploration with the user, alternating between observation of and discussion f its structure and details.

Interpretation:

Much of the work of the designer is reading between the lines of facts gathered about user's behaviors, their environment, and what they say. These facts must be taken together as a whole, and analyzed by the designer to uncover the design implications. Interviewers must be careful, however, to avoid assumptions based on their own interpretation of the facts without verifying these assumptions with users.

Focus:

Rather than coming to interviews with a set questionnaire or letting the interview wander aimlessly, the designer needs to subtly direct the interview so as to capture data relevant t design issues.

Improving on contextual inquiry

Contextual inquiry forms a solid theoretical foundation for quantitative research, but as a specific method it has some limitations and inefficiencies. The following process improvements result in a more highly leveraged research phase that better set the stage for successful design:

- **Shortening the interview process:** contextual inquiry assumes full day interviews with users. The authors have found that interviews as short as one hour in duration are sufficient to gather the necessary user data, provided that a sufficient number of interviews (about six well-selected users for each hypothesized role or type) are scheduled. It is much easier and more effective to find a diverse set of users who will consent to an hour with a designer than it is to find users who will agree to spend an entire day.
- **Using smaller design teams:** Contextual inquiry assumes a large design team that conducts multiple interviews in parallel, followed by debriefing sessions in which the full team participates. Experiments show that it is more effective to conduct interviews sequentially with the same designers in each interview. This allows the design team to remain small (two or three designers), but even more important, it means that the entire team interacts with all interviewed users directly; allowing the members to most effectively analyzed and synthesized the user data.
- **Identifying goals first:** Contextual inquiry, as described by Beyer and Holtzblatt, feeds a design process that is fundamentally task-focused. It is proposed that ethnographic interviews first identify and prioritize user goals before determining the tasks that relate to these goals.
- **Looking beyond business contexts:** the vocabulary of contextual inquiry assumes a business product and a corporate environment. Ethnographic interviews are also possible in consumer domains, though the focus of questioning is somewhat different.

© Copyright Virtual University of Pakistan

20.3 Preparing for ethnographic interviews

As we discussed ethnography is term borrowed form anthropology, meaning the systematic and immersive study of human cultures. In anthropology, ethnographic researchers spend years living immersed in the cultures they study and record. Ethnographic interviews take the spirit of this type of research and apply it on a micro level. Rather than trying to understand behaviors and social ritual of an entire culture, the goal is understand the behaviors and rituals of people interacting with individual products.

Identifying candidates

Because the designer must capture an entire range of user behaviors regarding a product, it is critical that the designers identify and appropriately diverse sample of users and user types when planning a series of interviews. Based on information gleaned form stakeholders, SMEs, and literature reviews, designers need to create a hypothesis that serves as a starting point I determining what sorts of users and potential users to interview.

Kim Goodwin has coined this the persona hypothesis, because it is the first step towards identifying and synthesizing personas. The persona hypothesis is based on likely behavioral differences, not demographics, but takes into consideration identified target markets and demographics. The nature of the product's domain makes a significant difference in how a persona hypothesis is constructed. Business users are often quite different than consumer users in their behavior patterns and motivations, and different techniques are used to build the persona hypothesis in each case.

The personal hypothesis

The persona hypothesis is a first cut at defining the different kinds of users (and sometimes customers) for a product in a particular domain. He hypothesis serves as a basis for an initial set of interviews; as interviews proceed, new interviews may be required if the data indicates the existence of user types not originally identified.
The persona hypothesis attempts to address, at a high level, these three questions:

- What different sorts of people might use this product?
- How might their needs and behaviors vary?
- What ranges of behavior and types of environments need to be explored?

Roles in business and customer domains

Patterns of needs and behavior, and therefore types of users, vary significantly between business and technical, and consumer products. For business products, roles—common sets of tasks and information needs related to distinct classes of users—provide an important initial organizing principle. For example, in an enterprise portal, these search roles can be found:

- People who search for content on the portal
- People who upload and update content on the portal
- People who technically administer the portal

© Copyright Virtual University of Pakistan

In business and technical context, roles often map roughly to job descriptions, so it is relatively easy to get a reasonable first cut of user types to interview by understanding the kind of jobs held by users of the system.

Unlike business users, consumers don't have concrete job descriptions, and their use of products tends to cross multiple contexts. Their roles map more closely to lifestyle choices, and it is possible for consumer users to assume multiple roles even for a single product in this sense. For consumers, roles can usually better be expressed by behavioral variables

Behavioral and demographic variables

Beyond roles, a persona hypothesis seeks to identify variables that might distinguish users based on their needs and behaviors. The most useful, but most difficult to anticipate without research, are behavioral variables: types of behavior that behavior concerning shopping that we might identify:

- Frequency of shopping (frequent--infrequent)
- Desire t shop (loves to shop—hates to shop)
- Motivation to shop (bargain hunting—searching for just the right item)

Although consumer user types can often be roughly defined by the combination of behavioral variables they map to, behavioral variables are also important for identifying types of business and technical users. People within a single business-role definition may have different motivations for being there and aspirations for what they plan to do in the future. Behavioral variables can capture this; through usually not until user data has been gathered.

Given the difficulty in accurately anticipating behavioral variables before user data is gathered, another helpful approach in building a persona hypothesis is making use of demographic variables. When planning your interviews, you can use market research to identify ages, locations, gender, and incomes of the target markets for the product. Interviews should be distributed across these demographic ranges.

Domain expertise versus technical expertise

One important type of behavioral distinction to note is the difference between technical expertise (knowledge of digital technology) and domain expertise (knowledge of a specialized subject area pertaining to a product). Different users will have varying amount of technical expertise; similarly, some users of a product may be less expert in their knowledge of the product's domain (for example, accounting knowledge in the case of a general ledger application). Thus, depending on who the design target of the product is, domain support may be a necessary part of the product's design, as well as technical ease of use.

Environmental variables

A final consideration, especially in the case of business products, is the cultural differences between organizations in which the users are employed. Small companies, for example, tend to have more interpersonal contact between workers: huge companies have layers of bureaucracy. These environmental variables also fall into ranges:

© Copyright Virtual University of Pakistan

- Company size (small—multinational)
- IT presence (ad hoc—draconian)
- Security level (lax--tight)

Like behavioral variables, these may be difficult to identify without some domain research, because patterns do vary significantly by industry and geographic region.

20.4 Putting a plan together

After you have created a persona hypothesis, complete with potential roles, behavioral, demographic, and environmental variables, you then need to create an interview plan that can be communicated to the person in charge of providing access to users.

Each identified role, behavioral variable, demographic variable, and environmental variable identified in the persona hypothesis should be explored in four to six interviews (some time more if a domain is particular complex). However, these interviews can overlap: it is perfectly acceptable to interview a female in her twenties who loves to shop; this would count as an interview for each of three different variables: gender, age group, and desire to shop. By being clever about mapping variables to interviewee screening profiles, you can keep the number of interviews to a manageable number.

© Copyright Virtual University of Pakistan

Lecture 21.

User Research Part-III

Learning Goals

The aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand how to conduct ethnographic interviews
- Discuss briefly other research techniques

21.1 How to conducting ethnographic interviews

Securing interviews

Use the followings to get access

1. Stakeholders
2. Market or usability research firm
3. Friends and relatives

Interview teams and timings
-2 interviewees, 1-hr per interview, 6 per day

21.2 Phases of ethnographic interviews

Begin	→	End
Structural issues	→	Specific issues
Goal-oriented issues	→	Task-oriented issues

Early-phase

- Exploratory
- Focused on domain knowledge
- Open-ended questions

Mid-phase

- Identify patterns of use
- Clarifying questions
- More focused questions

Late-phase

- Confirm patterns of use
- Clarify user roles and behaviors
- Closed-ended questions

21.3 Basic interview methods

- Interview where the action happens
- Avoid a fixed set of questions
- Focus on goals first, tasks second

© Copyright Virtual University of Pakistan

- Avoid making the user a designer
- Avoid discussions of technology
- Encouraging storytelling
- Ask for a show-and-tell
- Avoid leading questions

Basic interview methods

1. Goal-oriented questions
2. System-oriented questions
3. Workflow-oriented question
4. Attitude-oriented questions

1. Goal-oriented questions

Opportunity
What activities currently waste your time?

Goals

What makes a good day?
A bad day.

Priorities
What is the most important to you?

Information

What helps you make decisions?

2. System-oriented questions

Function
What are the most common things you do with the product?

Frequency

What parts of the product do you use most?

Preference

What are your favorite aspects of the product? What drives you crazy?

Failure

How do you work around problems?

Expertise

What shortcuts do you employ?

3. Workflow-oriented questions

Process
What did you do when you first came into today? And after that?

Occurrence and recurrence

How often do you do this?
What things do you do weekly, monthly but not every day?

© Copyright Virtual University of Pakistan

Exceptions
What constitutes a typical day?
What would be an unusual event?

4. Attitude-oriented questions
Aspiration
What do you see yourself doing five years from now?

Avoidance
What would you prefer not to do?
What do you procrastinate on?

Motivation
What do you enjoy most about your job (or lifestyle)?
What do you always tackle first?

21.4 Types of qualitative research

Stakeholder interview
Subject matter experts (SME) interviews
User and customer interviews
Literature review
Product/prototype and competitive audits
User observation/ethnographic field studies

21.5 Others types of research

1. Focus group
2. Market demographics and segments
3. Usability and user testing

1. Focus group

- Used by marketing organizations
- Used in traditional product marketing
- Representative users gathered in room
- Shown a product and reactions gauged
- Reactions recorded by audio/video

Limitations

2. Market demographics and segments
What motivates people to buy?

- Determined by market segmentation that group people by distinct needs
- Determines who will be receptive to what marketing message or a particular product
- Demographic data
  Race, education, income, location etc.
- Psychographic data

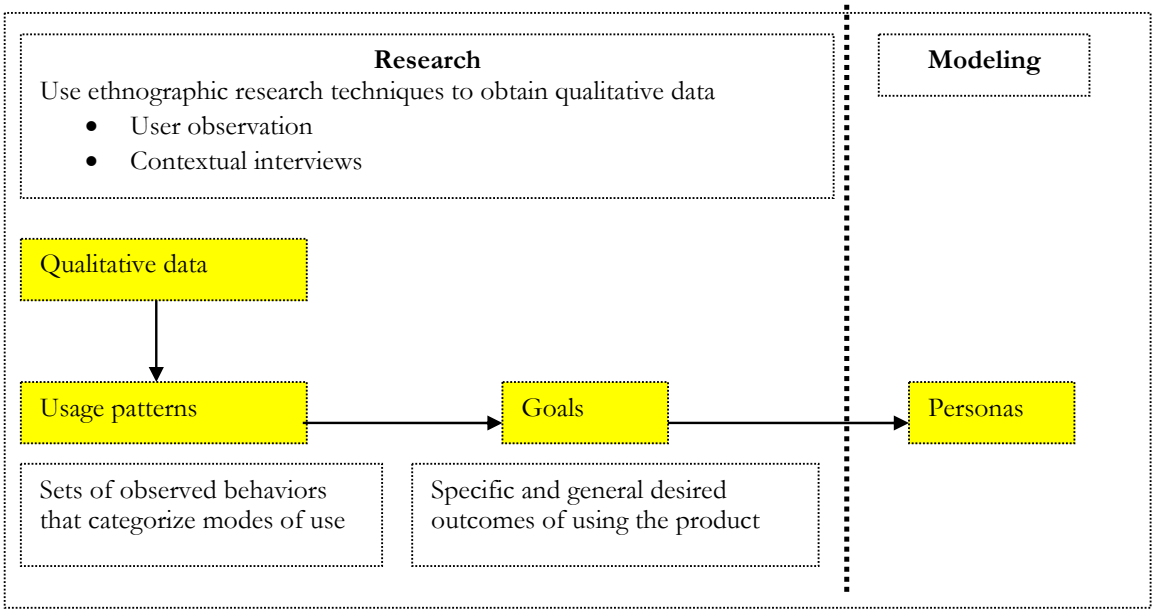
© Copyright Virtual University of Pakistan

Attitude, lifestyle, values, ideology etc.

3. Usability and user testing

21.6 Comparison of different techniques

Techniques	Good for	Kind of data	Advantages	Disadvantages
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee if necessary. Encourages contact between developers and users	Time consuming Artificial environment may intimidate interviewee
Studying documentation	Learning about procedures, regulations and standards	Quantitative	No time commitment from users required	Day-to-day working will differ from documented procedures
Naturalistic observation	Understanding context of user activity	Quantitative	Observing actual work gives insights that other techniques can't give	Very time consuming. Huge amounts of data
Focus groups and workshops	Collecting multiple view points	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters



© Copyright Virtual University of Pakistan

Lecture 22.

User Modeling

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand how to conduct ethnographic interviews
- Discuss briefly other research techniques

The most powerful tools are simple in concept, but must be applied with some sophistication. The most powerful interaction design tool used is a precise descriptive model of the user, what he wishes t accomplish, and why. The sophistication becomes apparent in the way we construct and use that model.

These user models, which we call personas, are not real people, but they are based on the behaviors and motivations of real people and represent them throughout the design process. They are composite archetypes based on behavioral data gathered from many actual users through ethnographic interviews. We discover out personas during the course of the Research phase and formalize them in the Modeling phase, by understanding our personas, we achieve and understanding of our users' goals in specific context—a critical tool for translating user data into design framework.

There are many useful models that can serve as tools for the interaction designer, but it is felt that personas are among the strongest.

22.1 Why Model?

Models are used extensively in design, development, and the sciences. They are powerful tools for representing complex structures and relationships for the purpose of better understanding or visualizing them. Without models, we are left to make sense of unstructured, raw data, without the benefit of the pig picture or any organizing principle. Good models emphasize the salient features of the structures or relationships they represent and de-emphasize the less significant details.

Because we are designing for users, it is important that we can understand and visualize the salient aspects of their relationships with each other, with their social and physical environment and of course, with the products we hope to design.

Just as physicists create models of the atom based on raw, observed data and intuitive synthesis of the patterns in their data, so must designers create models of users based on raw, observed behaviors and intuitive synthesis often patterns in the data. Only after we formalize such patterns can we hope to systematically construct patterns of interactions that smoothly match the behaviors, mental models and goals of users. Personas provide this formalization.

© Copyright Virtual University of Pakistan

22.2 Personas

To create a product that must satisfy a broad audience of users, logic tells you to make it as broad in its functionality as possible to accommodate the most people. This logic, however, is flawed. The best way to successfully accommodate a variety of users is to design for specific types of individuals with specific needs.

When you broadly and arbitrarily extend a product's functionality to include many constituencies, you increase the cognitive load and navigational overhead for all users. Facilities that map please some users will likely interfere with the satisfaction of other.
A simple example of how personas are useful is shown in figure below, if you try to design an automobile that pleases every possible driver, you end up with a car with every possible feature, but which pleases nobody. Software today is too often designed to please to many users, resulting in low user satisfaction



But by designing different cars for different people with different specific goals, as shown in figure below, we are able to create designs that other people with similar needs to our target drivers also find satisfying. The same hold true for the design of digital products and software.



The key is in choosing the right individuals to design for, ones whose needs represent the needs of a larger set of key constituents, and knowing how to prioritize design elements to

© Copyright Virtual University of Pakistan

address the needs of the most important users without significantly inconveniencing secondary users. Personas provide a powerful tool for understanding user needs, differentiating between different types of users, and prioritizing which users are the most important to target in the design of function and behavior.

Personas were introduced as a tool for user modeling, they have gained great popularity in the usability community, but they have also been the subjects of some misunderstandings.

Strengths of personas as a design tool

The persona is a powerful, multipurpose design tool that helps overcome several problems that currently plague the development of digital products. Personas help designers"

- Determine what a product should do and how it should behave. Persona goals and tasks provide the basis for the design effort.
- Communicate with stakeholders, developers, and other designers. Personas provide a common language for discussing design decisions, and also help keep the design centered on users at every step in the process.
- Build consensus and commitment to the design. With a common language comes a common understanding. Personas reduce the need for elaborate diagrammatic models because, as it is found, it is easier to understand the many nuances of user behavior through the narrative structures that personas employ.
- Measure the design's effectiveness. Design choices can be tested on a persona in the same way that they can be show to a real user during the formative process. Although this doesn't replace the need to test on real users. It provides a powerful reality check tool for designers trying to solve design problems. This allows design iteration to occur rapidly and inexpensively at the whiteboard, and it results in a far stronger design baseline when the time comes to test with real users.
- Contribute to other product-related efforts such as marketing and sales plan. It has been seen that clients repurpose personas across their organization, informing marketing campaigns, organizational structure, and other strategic planning activities. Business units outside of product development desire sophisticated knowledge of a product's users and typically view personas with great interest.

Personas and user-centered design

Personas also resolve three User-Centered design issues that arise during product development:

- The elastic user
- Self-referential design
- Design edge cases

The elastic user

Although satisfying the user is goal, the term user causes trouble when applied to specific design problems and contexts. Its imprecision makes it unusable as a design tool—every person on a product team has his own conceptions of the user and what the user needs. When it comes time to make a product decisions, this "user" becomes elastic, bending and stretching to fit the opinions and presuppositions of whoever has the floor.

© Copyright Virtual University of Pakistan

If programmers find it convenient to simply drop a user into a confuting file system of nested hierarchical folders to find the information she needs, they define the elastic user as an accommodating, computer-literate power user. Other times, when they find it more convenient to step the user through a difficult process with a wizard, they define the elastic user as an unsophisticated first-time user. Designing for the elastic user gives the developer license to code as he pleases while still apparently serving "the user". However, our goal is to design software that properly meets real user needs. Real users—and the personas representing them—are not elastic, but rather have specific requirements based on their goals, capabilities, and contexts.

Self-referential design

Self-referential design occurs when designers or developers project their own goals, motivations, skills, and mental models onto a product's design. Most "cool" product designs fall into this category: the audience doesn't extend beyond people like the designer, which is fine for a narrow range of products and completely inappropriate for most others. Similarly, programmers apply self-referential design when they create implementation-model products. They understand perfectly how it works and are comfortable with such products. Few non-programmers would concur.

Design edge cases

Another syndrome that personas help prevent is designing for edge cases—those situations that might possibly happen, but usually won't for the target personas. Naturally, edge cases must be programmed for, but they should never be the design focus. Personas provide a reality check for the design.

Personas are based on research

Personas must, like any model, be based on real-world observation. The primary source of data used to synthesize personas must be from ethnographic interviews, contextual inquiry, or other similar dialogues with and observation of actual and potential users. Other data that can support and supplement the creation of personas include, in rough order of efficacy:

- Interviews with users outside of their use contexts
- Information about users supplied by stakeholders and subject matter experts
- Market research data such as focus groups and surveys
- Market segmentation models
- Data gathered from literature reviews and previous studies

However, none of this supplemental data can take the place of direct interaction with and observation of users in their native environments. Almost every word in a well-developed persona's description can be traced back to user quotes or observed behaviors.

Personas are represented as individuals

Personas are user models that are represented as specific, individual humans. They are not actual people, but are synthesized directly from observations of real people. One of the key elements that allow personas to be successful as user models is that they are personifications. They are represented as specific individuals. This is appropriate and effective because of the unique aspects of personas as user models: they engage the

© Copyright Virtual University of Pakistan

empathy of the development team toward the human target of design. Empathy is critical for the designers, who will be making their decisions for design frameworks and details based on both the cognitive and emotional dimensions of the persona, as typified by the persona's goals. However, the power of empathy should not be quickly discounted for other team members.

Personas represent classes of users in context

Although personas are represented as specific individuals, at the same time they represent a class or type of user of a particular interactive product. Specifically, persona encapsulates a distinct set of usage patterns, behavior patterns regarding the use of a particular product. These patterns are identified through an analysis of ethnographic interviews, supported by supplemental data if necessary or appropriate. These patterns, along with work or lifestyle-related roles define personas as user archetype. Personas are also referred as composite user archetypes because personas are in sense composites assembled by clustering related usage patterns observed across individuals in similar roles during the research phase.

Personas and reuse

Organizations with more than one product often want to reuse the same personas. However, to be effective, personas must be context-specific—they should be focused on the behaviors and goals related to the specific domain of a particular product. Personas, because they are constructed from specific observations of users interacting with specific products in specific contexts, cannot easily be reused across products even when those products form a closely linked suite. Even then, the focus of behaviors may be quite different in one product than in another, so researchers must take care to perform supplemental user research.

Archetypes versus stereotype

Don't confuse persona archetype with stereotypes. Stereotypes are, in most respects, the antithesis of well-developed personas. Stereotypes represent designer or researcher biases and assumptions, rather than factual data. Personas developed drawing on inadequate research run the risk of degrading to stereotypical caricatures. Personas must be developed and treated with dignity and respect for the people whom they represent. Personas also bring to the forefront issues of social and political consciousness. Because personas provide a precise design target and also serve as a communication tool to the development team, the designer much choose particular demographic characteristics with care. Personas should be typical and believable, but not stereotypical.

Personas explore ranges of behavior

The target market for a product describes demographics as well as lifestyle and sometimes job roles. What it does not describe are the ranges of different behaviors that members of that target market exhibit regarding the product itself and product-related contexts. Ranges are distinct from averages: personas do not seek to establish an average user, but rather to identify exemplary types of behaviors along identified ranges.

Personas fill the need to understand how users behave within given product domain—how they think about it and what they do with it—as well as how they behave in other contexts that may affect the scope and definition of the product. Because personas must describe ranges f behavior to capture the various possible ways people behave with the

© Copyright Virtual University of Pakistan

product, designers must identify a collection or cast of personas associated with any given product.

Personas must have motivations

All humans have motivations that drive their behaviors; some are obvious, and many are subtle. It is critical that personas capture these motivations in the form of goals. The goals we enumerate for our personas are shorthand notation for motivations that not only point at specific usage patterns, but also provide a reason why those behaviors exist. Understanding why a user performs certain tasks gives designers great power to improve or even eliminate those tasks, yet still accomplish the same goals.

Personas versus user roles

User roles and user profiles each share similarities with personas; that is, they both seek to describe relationships of users to products. But persona and the methods by which they are employed as a design tool differ significantly from roles and profiles in several key aspects.

User roles or role models, are an abstraction, a defined relationship between a class of users and their problems, including needs, interests, expectations, and patterns of behavior. Holtzblatt and Beyer's use of roles in consolidated flow, cultural, physical, and sequence models is similar in that it attempts to isolate various relationships abstracted from the people possessing these relationships.
Problem with user role
There are some problems with user roles:

- It is more difficult to properly identify relationships in the abstract, isolated from people who posses them—the human power of empathy cannot easily be brought to bear on abstract classes of people.
- Both methods focus on tasks almost exclusively and neglect the use of goals as an organizing principle for design thinking and synthesis.
- Holzblatt and Beyer's consolidated models, although useful and encyclopedic in scope, are difficult to bring together as a coherent tool for developing, communicating, and measuring design decisions.

Personas address each of these problems. Well-developed personas incorporate the same type of relationships as user roles do, but express them in terms of goals and examples in narrative.

Personas versus user profile

Many usability parishioners use the terms persona and user profile synonymously. There is no problem with this if the profile is truly generated from ethnographic data and encapsulates the depth of information. Unfortunately, all too often, it has been seen that user profile =s that reflect Webster's definition of profile as a 'brief biographical sketch." In other words, user profiles are often a name attached to brief, usually demographic data, along with a short, fictional paragraph describing the kind of car this person drives, how many kids he has, where he lives, and what he does for a living. This kind of user profile is likely to be a user stereotype and is not useful as a design tool. Personas, although has names and sometimes even cars and family members, these are employed sparingly as narrative tools to help better communicate the real data and are not ends in themselves.

© Copyright Virtual University of Pakistan

Personas versus market segments

Marketing professionals may be familiar with a process similar to persona development because it shares some process similarities with market definition. The main difference between market segments and design personas are that the former are based on demographics and distributed channels, where as the latter are based on user behaviors and goals. The two are not the same and don't serve the same purpose. The marketing personas shed light on the sales process, whereas the design personas shed light on the development process. This said, market segments play role in personas development.

User personas versus non-user personas

A frequent product definition error is to target people who review, purchase, or administer the product, but who are not end users. Many products are designed for columnists who review the product in consumer publications. IT managers who purchase enterprise products are, typically, not the users the products. Designing for the purchaser is a frequent mistake in the development of digital products. In certain cases, such as for enterprise systems that require maintenance and administrator interface; it is appropriate to create non-user personas. This requires that research be expanded to include these types of people.

22.3 Goals

If personas provide the context for sets of observed behaviors, goals are the drivers behind those behaviors. A persona without goals can still serve as a useful communication tool, but it remains useless as a design tools. User goals serve as a lens through which designers must consider the functions of a product. The function and behavior of the product must address goals via tasks—typically as few tasks as absolutely necessary.

Goals motivate usage patterns

People's or personas' goals motivate them to behave the way they do. Thus, goals provide not only answer to why and how personas desire to use a product, but can also serve as a shorthand in the designer's mind for the sometimes complex behaviors in which a persona engages and, therefore, for the tasks as well.

Goals must be inferred from qualitative data

You can't ask a person what his goals are directly: Either he won't be able to articulate them, or he won't be accurate or even perfectly honest. People simply aren't well prepared to answer such questions accurately. Therefore, designers and researchers need to carefully reconstruct goals from observed behaviors, answers to other questions, non-verbal cues, and clues from the environment such as book titles on shelves. One of the most critical tasks in the modeling of personas is identifying goals and expressing them succinctly: each goal should be expressed as a simple sentence.

22.4 Types of goals

Goals come in many different verities. The most important goals from a user-centered design standpoint are the goals of users. These are, generally, first priority in a design, especially in the design of consumer products. Non-user goals can also come into play, especially in enterprise environments. The goals of organizations, employers, customers,

© Copyright Virtual University of Pakistan

and partners all need to be acknowledged, if not addressed directly, by the product's design.

User goals

User personas have user goals. These range from broad aspirations to highly pragmatic product expectations. User goals fall into three basic categories

- Life goals
- Experience goals
- End goals

Life goals

Life goals represent personal aspirations of the user that typically go beyond the context of the product being designed. These goals represent deep drives and motivations that help explain why the user is trying to accomplish the end goals he seeks to accomplish. These can be useful in understanding the broader context or relationships the user may have with others and her expectations of the product from a brand perspective.

Examples:

- Be the best at what I do
- Get onto the fast track and win that big promotion
- Learn all there is to know about this field
- Be a paragon of ethics, modesty and trust

Life goals rarely figure directly into the design of specific elements of an interface. However, they are very much worth keeping in mind.

Experience goals

Experience goals are simple, universal, and personal. Paradoxically, this makes them difficult for many people to talk about, especially in the context of impersonal business. Experience goals express how someone wants to feel while using a product or the quality of their interaction with the product.

Examples

- Don't make mistakes
- Feel competent and confident
- Have fun

Experience goals represent the unconscious goals that people bring to any software product. They bring these goals to the context without consciously realizing it and without necessarily even being able to articulate the goals.

End goals

End goals represent the user's expectations of the tangible outcomes of using specific product. When you pick op a cell phone, you likely have an outcome in mind. Similarly, when you search the web for a particular item or piece of information, you have some

© Copyright Virtual University of Pakistan

clear end goals to accomplish. End goals must be met for users to think that a product is worth their time and money, most of the goals a product needs to concern itself with are, therefore, end goals such as the following:

- Find the best price
- Finalize the press release
- Process the customer's order
- Create a numerical model of the business

Non-user goals

Customer goals, corporate goals, and technical goals are all non-user goals. Typically, these goals must be acknowledged and considered, but they do not form the basis for the design direction. Although these goals need to be addressed, they must not be addressed at the expense of the user.

Types of non-user goals

- Customer goals
- Corporate goals
- Technical goals

Customer goals

Customers, as already discussed, have different goals than users. The exact nature of these goals varies quite a bit between consumer and enterprise products. Consumer customers are often parents, relatives, or friends who often have concerns about the safety and happiness of the persons for whom they are purchasing the product. Enterprise customers are typically IT managers, and they often have concerns about security, ease of maintenance, and ease of customization.

Corporate goals

Business and other organizations have their own requirements for software, and they are as high level as the personal goals of the individual. "To increase our profit" is pretty fundamental to the broad of directors or the stockholders. The designers use these goals to stay focused on the bigger issues and to avoid getting distracted by tasks or other false goals.

Examples

- Increase profit
- Increase market share
- Defeat the competition
- Use resources more efficiently
- Offer more products or services

© Copyright Virtual University of Pakistan

Technical goals

Most of the software-based products we use everyday are created with technical goals in mind. Many of these goals ease the task of software creation, which is a programmer's goal. This is why they take precedence at the expense of the users' goals.

Example:

- Save money
- Run in a browser
- Safeguard data integrity
- Increase program execution efficiency

22.5 Constructing personas

Creating believable and useful personas requires an equal measure of detailed analysis and creative synthesis. A standardized process aids both of these activities significantly.
Process of constructing personas involve following steps:

1. Revisit the persona hypothesis
2. Map interview subjects to behavioral variables
3. Identify significant behavior patterns
4. Synthesize characteristics and relevant goals.
5. Check for completeness.
6. Develop narratives
7. Designate persona types

Revisit the persona hypothesis

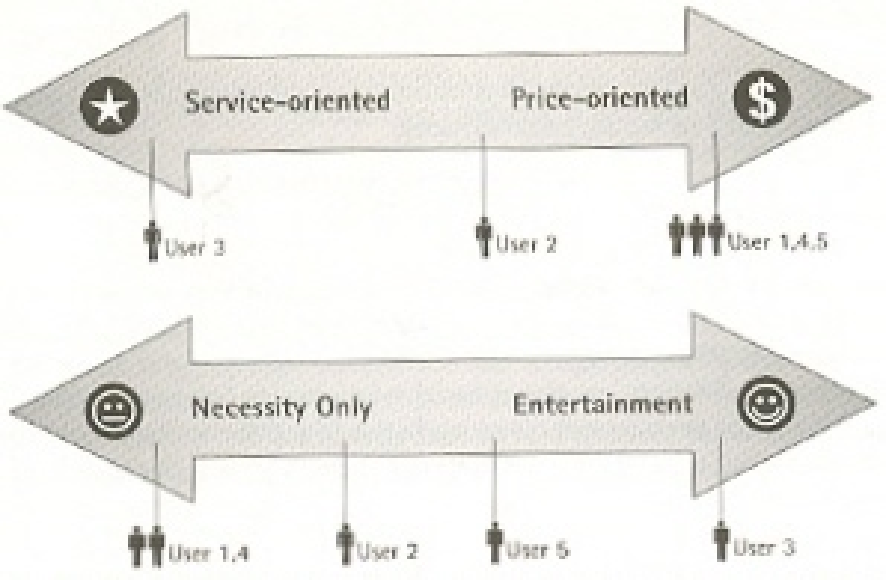
After you have completed your research and performed a cursory organization of the data, you next compare patterns identified in the data to the assumptions make in the persona hypothesis. Were the possible roles that you identified truly distinct? Were the behavioral variables you identified valid? Were there additional, unanticipated ones, or ones you anticipated that weren't supported by data?

If your data is at variance with your assumptions, you need to add, subtract, or modify the roles and behaviors you anticipated. If the variance is significant enough, you may consider additional interviews to cover any gaps in the new behavioral ranges that you've discovered.

Map interview subjects to behavioral variables

After you are satisfied that you have identified the entire set f behavioral variables exhibited by your interview subjects, the next step is to map each interviewee against each variable range that applies. The precision of this mapping isn't as critical as identifying the placement f interviewees in relationship to each other. It is the way multiple subjects cluster on each variable axis that is significant as show in figure.

© Copyright Virtual University of Pakistan



Identify significant behavior patterns

After you have mapped your interview subjects, you see clusters of particular subjects that occur across multiple ranges or variables. A set of subjects who cluster in six to eight different variables will likely represent a significant behavior patterns that will form the basis of a persona. Some specialized role may exhibit only one significant pattern, but typically you will find two or even three such patterns. For a pattern to be valid, there must be a logical or causative connection between the clustered behaviors, not just a spurious correlation.

Synthesize characteristic and relevant goals

For each significant behavior pattern you identify, you must synthesize details from your data. Describe the potential use environment, typical workday, current solutions and frustrations, and relevant relationships with other.

Brief bullet points describing characteristics of the behavior are sufficient. Stick to observed behaviors as mush as possible; a description or two that sharpen the personalities of your personas and help bring them to life.

One fictional detail at this stage is important: the persona's first name and last names. The name should be evocative of the type of person the persona is, without tending toward caricature or stereotype.

Goals are the most critical detail to synthesize from your interviews and observations of behaviors. Goals are best derived from an analysis of the group of behaviors comprising each persona. By identifying the logical connections between each persona's behaviors, you can begin to infer the goals that lead to those behaviors. You can infer goals both by observing actions and by analyzing subject responses to goal-oriented interview questions.

© Copyright Virtual University of Pakistan

Develop narratives

Your list of bullet point characteristics and goals point to the essence of complex behaviors, but leaves much implied. Third-person narrative is far more powerful in conveying the persona's attitudes, needs, and problems to other team members. It also deepens the designer's connection to the personas and their motivations.

A typical narrative should not be longer than one or two pages of prose. The narrative must be nature, contain some fictional events and reactions, but as previously discussed, it is not a short story. The best narrative quickly introduces the persona in terms of his job or lifestyle, and briefly sketches a day in his life, including peeves, concerns, and interests that have direct bearing on the product.

Be careful about precision of detail in your descriptions. The detail should not exceed the depth of your research. When you start developing your narrative, choose photographs f your personas. Photographs make them feel more real as you create the narrative and engage others on the team when you are finished.

Designate persona types

By now your personas should feel very much like a set of real people that you feel you know. The final step in persona construction finishes the process f turning your qualitative research into a powerful set of design tools.

There are six types of persona, and they are typically designated in roughly the ordered listed here:

- Primary
- Secondary
- Supplemental
- Customer
- Served
- Negative

Primary Persona:
The main user the product is built for; their needs must be fully met.
Secondary Persona:
A user who also uses the product, but not the main focus; their needs are considered if they don ' t conflict with the primary persona.
Supplemental Persona:
A user with different needs that may not be fully supported; their needs are only addressed if possible without affecting others.
Customer Persona:
The person who buys the product, but may not be the actual user.
Served Persona:
Someone affected by the product but doesn ' t use it directly (e.g., a patient ' s family for a healthcare app).
Negative Persona:
A type of user the product is not meant for; helps to avoid designing for the wrong audience.

Primary personas

Primary personas represent the primary target for the design of an interface. There can be only one primary persona per interface for a product, but it is possible for some products to have multiple distinct interfaces, each targeted at a distinct primary persona.

Secondary personas

Sometimes a situation arises in which a persona would be entirely satisfied by a primary persona's interface if one or two specific additional needs were addressed by the interface. This indicates that the persona in question is a secondary persona for that interface, and the design of that interface must address those needs without getting in the way of the primary persona. Typically, an interface will have zero to two secondary personas.

Supplemental personas

User personas that are not primary or secondary are supplemental personas: they are completely satisfied by one of the primary interface. There can be any number of

© Copyright Virtual University of Pakistan

supplemental personas associated with an interface. Often political personas—the one added to the cast to address stakeholder assumptions—become supplemental personas.

Customer persona

Customer personas address the needs of customers, not end users. Typically, customer personas are treated like secondary personas. However, in some enterprise environment, some customer personas may be primary personas for their own administrative interface.

Served personas

Served personas are somewhat different from the persona types already discussed. They are not users of the product at all; however, they are directly affected by the use of the product. Served personas provide a way to track second-order social and physical ramifications of products. These are treated like secondary personas.

Negative personas

Like served personas, negative personas aren't users of the product. Unlike served personas, their use is purely rhetorical, to help communicate to other members of the team who should definitely not be the design target for the product. Good candidates for negative personas are often technology-savvy early-adopter personas for consumer products and IT specialists for end-user enterprise products.

© Copyright Virtual University of Pakistan

Lecture 23.

Requirements

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the narratives and scenarios
- Define requirements using persona-based design

It has already been discussed how to capture qualitative information about users. Through careful analysis of this information and synthesis of user models, we can get a clear picture of our users and their respective goals. It has also been explained how to prioritize which users are the most appropriate design targets. The missing piece to the puzzle, then, is the process of translating this knowledge into coherent design solutions that meet the needs of users while simultaneously addressing business needs and technical constraints.
Now we shall describe a process for bridging the research-design gap. It employs personas as the main characters in set of techniques that rapidly arrive at design solutions in an iterative repeatable and testable fashion. This process has three major milestones: defining user requirements; using these requirements to in turn define the fundamental interaction framework for the product; and filling in the framework with ever-increasing amounts of design detail. The glue that holds the process together is narrative: use of personas to tell stories that point to design.

23.1 Narrative as a design tool

Narrative, or storytelling, is one of the oldest human activities. Much has been written about the power of narrative to communicate ideas. However, narrative can also, through its efficacy at engaging and stimulating creative visualization skills, serve as a powerful tool in generating and validating design ideas. Because interaction design is first and foremost the design of behavior that occurs over time, a narrative structure, combined with the support of minimal visualization tools such as the whiteboard, is perfectly suited for envisioning and representing interaction concept. Detailed refinement calls for more sophisticated visual and interactive tools, but the initial work of defining requirements and frameworks is best done fluidly and flexibly, with minimal reliance on technologies that will inevitably impede ideation.

Scenarios in design

Scenario is a term familiar to usability professional, commonly used to describe a method of design problem solving by concretization: making use of a specific story to both construct and illustrate design solutions. Scenarios are anchored in the concrete, but permit fluidity; any member of the design team can modify them at will. As Carroll states in his book, Making Use:

> Scenarios are paradoxically concrete but rough, tangible but flexible … they implicitly encourage ‘what-if’? Thinking among all parties. They permit the articulation of design possibilities without understanding innovation …. Scenarios

© Copyright Virtual University of Pakistan

compel attention to the use that will be made of the design product. They can describe situations at many levels of detail, for many different purposes, helping to coordinate various aspects of the design project.

Carroll's use of scenario-based design focuses on describing how users accomplish tasks. It consists of an environment setting and includes agents or actors that are abstracted stand-ins for users, with role-based names such as Accountant or Programmer.
Although Carroll certainly understands the power and importance of scenarios in the design process, there can be two problems with scenarios as Carroll approaches them:

- Carroll's scenarios are not concrete enough in their representation of the human actor. It is impossible to design appropriate behaviors for a system without understanding in specific detail the users of the system. Abstracted, role-oriented models are not sufficient concrete to provide understanding or empathy with users.
- Carroll's scenarios jump too quickly to the elaboration of tasks without considering the user's goals and motivations that drive and filter these tasks. Although Carroll does briefly discuss goals, he refers only to goals of the scenario. These goals are somewhat circularly defined as the completion of specific tasks. Carroll's scenarios begin at the wrong level of detail: User goals need to be considered before user tasks can be identified and prioritized. Without addressing human goals, high-level product definition becomes difficult.

The missing ingredient in scenario-based methods is the use of personas. A persona provides a sufficiently tangible representation of the user to act as a believable agent in the setting of a scenario. This enhances the designer's ability to empathize with user mental models and perspectives. At the same time, it permits an exploration of how user motivations inflect and prioritize tasks. Because personas model goals and not simply tasks, the scope of the problem that scenarios address can also be broadened to include product definition. They help answer the questions, "what should this product be?" and "how this product should look and behave?"

Using personas in scenarios

Persona-based scenarios are concise narrative descriptions of one or more personas using a product to achieve specific goals. Scenarios capture the non-verbal dialogue between artifact and user over time, as well as the structure and behavior of interactive functions. Goals serve as filter for tasks and as guides for structuring the display of information and controls during the interactive process of constructing the scenarios.

Scenario content and context are derived from information gathered during the Research phase and analyzed during the modeling phase. Designers role-play personas as the characters in these scenarios, similar to actors performing improvisation. This process leads to real-time synthesis of structure and behavior—typically, at a whiteboard—and later informs the detailed look and feel. Finally, personas and scenarios are used to test the validity of design ideas and assumptions throughout the process. Three types of persona-based scenarios are employed at different points in the process, each time with a successively narrower focus.

© Copyright Virtual University of Pakistan

Persona-based scenarios versus use cases

Scenarios and use cases are both methods of describing a digital system. However, they serve very different functions. Goal-directed scenarios are an iterative means of defining the behavior of a product from the standpoint of specific users. This includes not only the functionality of the system, but the priority of functions and the way those functions are expressed in terms of what the user sees and how he interacts with the system.

Use cases, on the other hand, are a technique that has been adopted from software engineering by some usability professionals. They are usually exhaustive description f functional requirements of the system, often of a transactional nature, focusing on low-level user action and system responds—is not, typically, part of conventional or concrete use case, many assumptions about the form and behavior of the system to be designed remain implicit. Use cases permit a complete cataloguing of user tasks for different classes of users, but say little or nothing about how these tasks are presented to the user or how they should be prioritized in the interface. Use cases may be useful in identifying edge cases and for determining that a product is functionally complete, but they should be deployed only in the later stages of design validation.

23.2 Envisioning solutions with persona-based design

It has already been discussed that the translation from robust models to design solutions really consists of two major phases. Requirements Definition answers the broad questions about what a product is and what it should do, and Framework Definition answers questions about how a product behaves and how it is structured to meet user goals. Now we look Requirement Definition phase in detail.

Defining the requirements

The Requirement Definition phase determines the what of the design: what functions our personas need to use and what kind of information they must access to accomplish their goals. The following five steps comprise this process:

1. Creating problem and vision statement
2. Brainstorming
3. Identifying persona expectations
4. Constructing the context scenario
5. Identifying needs

Although these steps proceed in roughly chronological order, they represent an iterative process. Designers can expect to cycle through step 3 through 5 several times until the requirements are stable. This is a necessary part of the process and shouldn't be short-circuited. A detailed description of each of these steps follows.

Step1: Creating problem and vision statement

Before beginning any process of ideation, it's important for designers to have a clear mandate for moving forward, even if it is a rather high-level mandate. Problem and vision statements provide just such a mandate and are extremely helpful in building consensus among stakeholders before the design process moves forward.

At a high level, the problem statement defines the objective of the design. A design problem statement should concisely reflect a situation that needs changing, for both the

© Copyright Virtual University of Pakistan

personas and for the business providing the product to the personas. Often a cause-and-effect relationship exists between business concerns and persona concerns.

For example:

> Company X's customer satisfaction ratings are low and market share has diminished by 10% over the past year because users don't have adequate tools to perform X, Y and Z tasks that would help them meet their goal of G.

The connection of business issues to usability issues is critical to drive stakeholders' buy-in to design efforts and to frame the design effort in term of both user and business goals. The vision statement is an inversion of the problem statement that serves as a high-level design vision or mandate. In the vision statement, you lead with the user's needs, and you transition from those to how business goals are met by the design vision:

> The new design of Product X will help users achieve G by giving them the ability to perform X, Y and Z with greater [accuracy, efficiency, and so on], and without problems A, B, C that they currently experience. This will dramatically improve Company X's customer satisfaction ratings and leads to increased market share.

The content of both the problem and vision statement should come directly from research and user models. User goals and needs should derive from the primary and secondary personas, and business goals should be extracted from stakeholder interviews.

Step 2: Brainstorming

Brainstorming performed at this earlier stage of Requirements Definition assumes a somewhat ironic purpose. As designers, you may have been researching and modeling users and the domain for days or even weeks. It is almost impossible that you have not had design ideas percolating in your head. Thus, the reason we brainstorming at this point in the process is t get these ideas out our heads so we can "let them go" at least for the time being. This serves a primary purpose of eliminating as much designer bias as possible before launching into scenarios, preparing the designers to take on the roles of the primary personas during the scenario process.

Brainstorming should be unconstrained and critical—put all the wacky ideas you've been considering (plus some you haven't) out n the table and be the prepared to record them and file them away for safekeeping until much later in the process. It's not likely any of them will be useful in the end, but there might be the germ of something wonderful that will fit into the design framework you later create. Holtzblatt & Beyer describe a facilitated method for brainstorming that can be useful for getting a brainstorming session started, especially if your team includes non-designers.

Step 3: Identifying persona expectations

The expectations that your persona has for a product and its context of use is, collectively, that persona's mental model of the product. It is important that the representation model of the interface—how the design behaves and presents itself—should match the user's mental model as closely as possible, rather than reflecting the implementation model of how the product is actually constructed internally.

© Copyright Virtual University of Pakistan

For each primary persona you must identify:

- General expectations and desires each may have about the experience of using the product
- Behaviors each will expect or desire from the product
- Attitude, past experience, aspirations and other social, cultural, environmental and cognitive factors that influence these desires

Your persona descriptions may contain enough information to answer some of these questions directly; however, you should return to your research data to analyze the language and grammar of how user subjects and describe objects and actions that are part of their usage patterns. Some things to look for include:

- What do the subjects mention first?
- Which action words (verbs) do they use?
- Which intermediate steps, tasks, or objects in a process don't they mention?

After you have compiled a good list of expectations and influences, do the same for secondary and customer personas and crosscheck similarities and differences.

Step 4: constructing context scenarios

Scenarios are stories about people and their activities. Context scenarios are, in fact, the most story-like of the three types of scenario we employ in that the focus is very much on the persona, her mental models, goals, and activities. Context scenarios describe the broad context in which usage patterns are exhibited and include environmental and organizational considerations. Context scenarios establish the primary touch-points that each primary and secondary persona has with the system over the course of a day, o some other meaningful length of time that illuminates modes of frequent and regular use. Context scenarios are sometimes, for this reason, called day-in-the-life scenarios.
Context scenarios address questions such as the following

- What is the setting in which the product will be used?
- Will it be used for extended amounts or time?
- Is the persona frequently interrupted?
- Are there multiple users on a single workstation/device?
- What other products is it used with?
- How much complexity is permissible, based on persona skill and frequency of use?
- What primary activities does the persona need to accomplish to meet her goals?
- What is the expected end result of using the product?

To ensure effective context scenarios, keep them broad and relatively shallow in scope. Resist the urge of dive immediately into interaction detail. It is important to map out the pig picture first and systematically identify needs. Doing this and using the steps that follow prevent you from getting lost in design details that may not fit together coherently later.

© Copyright Virtual University of Pakistan

Context scenarios should not represent system behaviors as they currently are. These scenarios represent the brave new world of goal-directed products, so, especially in the initial phases, focus on the goals. Don't yet worry about exactly how things will get accomplished—you can initially treat the design as a bit of a magic black box. Sometimes more than one context scenario is necessary. This is true especially when there are multiple primary personas, but sometimes even a single primary persona may have two or more distinct contexts of use.

An example text scenario

The following is an example of a first iteration of a context scenario for a primary persona for a PDA/phone convergence device and service; Salman, a real-estate agent in Lahore. Salman's goals are to balance work and home life, cinch the deal, and make each client feel like he is his only client. Salman's context scenario might be as follow:

1. Getting ready in the morning, Salman uses his phone to check e-mail. It has a large enough screen and quick connection time so that it's more convenient than booting up a computer as he rushes to make his daughter, Alia, a sandwich for school.
2. Salman sees e-mail from his newest client who wants to see a house this afternoon. Salman entered his contact info a few days ago, so now he can call him with a simple action right from the e-mail.
3. While on the phone with his client, Salman switches to speakerphone so he can look at the screen while talking. He looks at his appointments to see when he's free. When he creates a new appointment, the phone automatically makes it an appointment with client, because it knows with whom he is talking. He
4. Quickly keys the address of the property into the appointment as he finishes his conversation.
5. After sending Alia off to school, Salman heads into the real-estate office to gather the papers he needs for the plumber working on another property. His phone has already updated his Outlook appointments so the rest of the office knows where he'll be in the afternoon.
6. The day goes by quickly, and he's running a bit late. As he heads towards the property he'll be showing client, the phone alerts him that his appointment is in 15 minutes. When he flips open the phone, it shows not only the appointment, but a list of all documents related to client, including e-mail, memos, phone messages, call logs to client's number, and even thumbnail pictures of the property that Salman sent as e-mail attachments. Salman presses the call button, and the phone automatically connects to client because it knows his appointment with him is soon. He lets him know he'll be there in 20 minutes.
7. Salman knows the address of the property, but is a bit unsure exactly where it is. He pulls over and taps the address he put into the appointment. The phone downloads directions along with a thumbnail map showing his location relative to the destination.
8. Salman gets to the property on time and starts showing it to client. He hears the phone ring from his pocket. Normally while he is in an appointment, the phone will automatically transfer directly to voicemail, but Alia has a code she can press to get through. The phone knows it's Alia calling, and uses a distinctive ring tone.
9. Salman takes the call—Alia missed the bus and needs a pickup. Salman calls her daughter and ask her that he will pick her up after 30 minutes.

© Copyright Virtual University of Pakistan

Now how the scenario remains at a fairly high level, not getting into too specific about interface or technologies. It's important to create scenarios that are within the realm of technical possibility, but at this stage the details of reality aren't yet important.

Step 5: identifying needs

After you are satisfied with an initial draft of your context scenario, you can begin to analyze it to extract the persona' needs. These needs consist of objects and actions as well as contexts. It is preferred not to think of needs as identical to tasks. The implication is that tasks must be manually performed by the user, whereas the term needs implies simply that certain objects need to exist and that certain actions on them need to happen in certain contexts. Thus, a need from the scenario above might be: Call (action) a person (object) directly from an appointment (context) If you are comfortable extracting needs in this format, it works quite well; you can separate them as described in the following sections.

Data needs

Persons' data needs are the objects and information that must be represented in the system. Charts, graphs, status markers, document types, attributes to be sorted, filtered, or manipulated, and graphical object types to be directly manipulated are examples of data needs.

Functional needs

Functional needs are the operations that need to be performed on the objects of the system and which are eventually translated into interface controls. Functional needs also define places or containers where objects or information in the interface must be displayed.

Contextual needs and requirements

Contextual needs describe relationships between sets of objects or sets of controls, as well as possible relationship between objects and controls. This can include which types of objects to display together to make sense for workflow or to meet specific persona goals, as well as how certain objects must interact with other objects and the skills and capabilities of the personas using the product.

Other requirements

It's important to get a firm idea of the realistic requirements of the business and technology you are designing.

- Business requirements can include development timelines, regulations, pricing structures, and business models.
- Technical requirements an include weight, size, form-factor, display, power constraints, and software platform choices.
- Customer and partner requirements can include ease of installation, maintenance, configuration, support costs, and licensing agreements.

Now design team should have a mandate in the form of the problem and vision statements, a rough, creative overview of how the product is going to address user goals in the form of context scenarios, and a reductive list of needs and requirements extracted from your research user models, and scenarios.

© Copyright Virtual University of Pakistan

Lecture 24.

Framework and Refinements

Learning Goals

The aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Discuss how to build an interaction framework?
- Discuss how to refine the form and behaviour?

24.1 Defining the interaction framework

The Requirements Definition phase sets the stage for the core of the design effort: defining the interaction framework of the product. The interaction framework defines not only the skeleton of the interaction — its structure — but also the flow and behavior of the product. The following six steps describe the process of defining the interaction framework:

1. Defining form factor and input methods
2. Defining views
3. Defining functional and data elements
4. Determining functional groups and hierarchy
5. Sketching the interaction framework
6. Constructing key path scenarios

Like previous processes, this is not a linear effort, but requires iteration. The steps are described in more detail in the following sections.

STEP 1: DEFINING FORM FACTOR AND INPUT METHODS

The first step in creating a framework is defining the form factor of the product you'll be designing. Is it a Web application that will be viewed on a high-resolution computer screen? Is it a phone that must be small, light, low-resolution, and visible in the dark and as well as in bright sunlight? Is it a kiosk that must be rugged to withstand a public environment with thousands of distracted, novice users? What are the constraints that each of these imply for any design? Answering these questions sets the stage for all subsequent design efforts.

After you have defined this basic *posture* of the product, you should then determine the valid input methods for the system: Keyboard, mouse, keypad, thumb-board, touch screen, voice, game controller, remote control, and many other possibilities exist Which combination is appropriate for your primary and secondary personas? What is the *primary* input method for the product?

© Copyright Virtual University of Pakistan

STEP 2: DEFINING VIEWS

The next step, after basic form factor and input methods are defined, is to consider which primary screens or states the product can be in. Initial context scenarios give you a feel for what these might be: They may change or rearrange somewhat as the design evolves (particularly in step 4), but it is often helpful to put an initial stake in the ground to serve as a means for organizing your thoughts. If you know that a user has several end goals and needs that don't closely relate to each other in terms of data overlap, it might be reasonable to define separate views to address them. On the other hand, if you see a cluster of related needs (for example, to make an appointment, you need to see a calendar and possibly contacts), you might consider defining a view that incorporates all these together, assuming the form factor allows it.

STEP 3: DEFINING FUNCTIONAL AND DATA ELEMENTS

Functional and data elements are the visible representations of functions and data in the interface. They are the concrete manifestations of the functional and data needs identified during the Requirements Definition phase. Where those needs were purposely described in terms of real-world objects and actions, functional and data elements are described in the language of user interface representations:

- Panes, frames, and other containers on screen
- Groupings of on-screen and physical controls
- Individual on-screen controls
- Individual buttons, knobs, and other physical affordances on a device
- Data objects (icons, listed items, images, graphs) and associated attributes

In early framework iterations, containers are the most important to specify; later as you focus on the design of individual containers, you will get to more detailed interface elements.

Many persona needs will spawn multiple interface elements to meet those needs. For example, Salman needs to be able to telephone his contacts. Functional elements to meet that need include:

- Voice activation (voice data associated with contact)
- Assignable quick-dial buttons
- Selecting from a list of contacts
- Selecting the name from e-mail header, appointment, or memo
- Auto-assignment of a call button in proper context (appointment coming up)

Multiple vectors are often a good idea, but sometimes not all possible vectors will be useful to the persona. Use persona goals, design principles, and patterns, as well as business and technical constraints to winnow your list of elements for meeting particular needs. You will also need to determine data elements. Some of Salman's data elements might include appointments, memos, to-do items, and messages.

STEP 4: DETERMINING FUNCTIONAL GROUPS AND HIERARCHY

After you have a good list of top-level functional and data elements, you can begin to group them into functional units and determine their hierarchy (Shneiderman, 1998).

© Copyright Virtual University of Pakistan

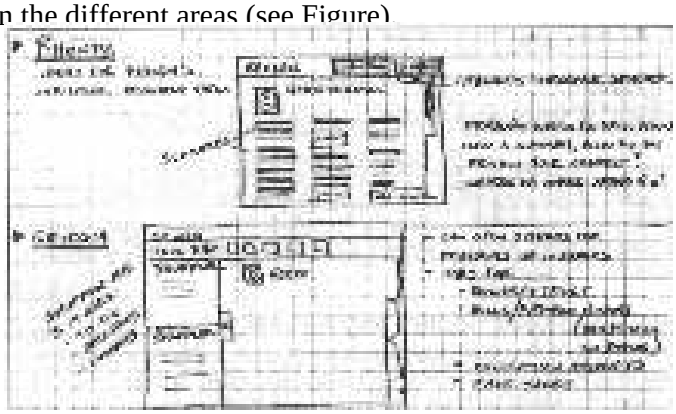
Because these elements facilitate specific tasks, the idea is to group elements to best facilitate the personal both within a task and between related tasks. Some issues to consider include:

- Which elements need a large amount of real estate and which do not?
- Which elements are *containers* for other elements?
- How should containers be arranged to optimize flow?
- Which elements are used together and which aren't?
- In what sequence will a set of related elements be used?
- What interaction patterns and principles apply?
- How do the personas' mental models affect organization? (Goodwin. 2002)

The most important initial step is determining the top-level container elements for the interface, and how they are best arranged given the form factor and input methods that the product requires. Containers for objects that must be compared or used together should be adjacent to each other. Objects representing steps in a process should, in general, be adjacent and ordered sequentially. Use of interaction design principles and patterns is extremely helpful at this juncture.

STEP 5: SKETCHING THE INTERACTION FRAMEWORK

You may want to sketch different ways of fitting top-level containers together in the interface. Sketching the framework is an iterative process that is best performed with a small, collaborative group of one or two interaction designers and a visual or industrial designer. This visualization of the interface should be extremely simple at first: boxes representing each functional group and/or container with names and descriptions of the relationships between the different areas (see Figure)



Be sure to look at the entire, top-level framework first; don't let yourself get distracted by the details of a particular area of the interface. There will be plenty of time to explore the design at the widget level and, by going there too soon, you risk a lack of coherence in the design later.

STEP 6: CONSTRUCTING KEY PATH SCENARIOS

Key path scenarios result from exploring details hinted at, but not addressed, in the context scenarios. Key path scenarios describe at the task level the primary actions and pathways through the interface that the persona takes with the greatest frequency, often on a daily basis. In an e-mail application, for example, viewing and composing mail are key path activities; configuring a new mail server is not.

© Copyright Virtual University of Pakistan

Key path scenarios generally require the greatest interaction support. New users must master key path interactions and functions quickly, so they need to be supported by built-in pedagogy. However, because these functions are used frequently, users do not remain dependent on that pedagogy for long: They will rapidly demand shortcuts. In addition, as users become very experienced, they will want to customize daily use interactions so that they conform to their individual work styles and preferences.

SCENARIOS AND STORYBOARDING

Unlike the goal-oriented context scenarios, key path scenarios are more task-oriented; focusing on task details broadly described and hinted at in the context scenarios (Kuutti, 1995). This doesn't mean that goals are ignored — goals and persona needs are the constant measuring stick throughout the design process, used to trim unnecessary tasks and streamline necessary ones. However, key path scenarios *must describe in exacting detail the precise behavior of each major interaction* and provide a walkthrough (Newman & Lamming, 1995) of each major pathway.

Typically, key path scenarios begin at a whiteboard and reach a reasonable level of detail. At some point, depending on the complexity and density of the interface, it becomes useful to graduate to computer-based tools. Many experts are fond of Microsoft PowerPoint as a tool for aiding in the storyboarding of key path scenarios. Storyboarding is a technique borrowed from filmmaking and cartooning. Each step in an interaction, whether between the user and the system, multiple users, or some combination thereof (Holtzblatt & Beyer, 1998) can be portrayed on a slide, and clicking through them provides a reality check for the coherence of the interaction (see Figure). PowerPoint is sufficiently fast and low-resolution to allow rapid drawing and iterating without succumbing to creating excessive detail.



PRETENDING THE SYSTEM IS HUMAN

Just as pretending it's magic is a powerful tool for constructing concept-level, context scenarios, pretending the system is human is a powerful tool at the interaction-level appropriate to key path scenarios. The principle is simple: Interactions with a digital system should be similar in tone and helpfulness to interactions with a polite, considerate human (Cooper. 1999). As you construct your interactions, you should ask yourself: Is the primary persona being treated humanely by the product? What would a thoughtful, considerate interaction look like? In what ways can the software offer helpful information without getting in the way? How can it minimize the persona’s effort in reaching his goals? What would a helpful human do?

© Copyright Virtual University of Pakistan

PRINCIPLES AND PATTERNS

Critical to the translation of key path scenarios to storyboards (as well as the grouping of elements in step 3) is the application of general interaction principles and specific interaction patterns. These tools leverage years of interaction design knowledge — not to take advantage of such knowledge would be tantamount to re-inventing the wheel. Key path scenarios provide an inherently top-down approach to interaction dsign, iterating through successively more-detailed design structures from main screens down to tiny subpanes or dialogs. Principles and patterns add a bottom-up approach to balance the process. Principles and patterns can be used to organize elements at all levels of the design.

24.2 Prototyping

It is often said that users can't tell you what they want, but when they see some thing and get to use it, they soon know what they don't want. Having collected information about work practices and views about what a system should and shouldn't do, we then need to try out our ideas by building prototypes and iterating through several versions. And the more iterations, the better the final product will be.

What is a prototype?

When you hear the term *prototype,* you may imagine something like a scale model of a building or a bridge, or maybe a piece of software that crashes every few minutes. But a prototype can also be a paper-based outline of a screen or set screens, an electronic "picture," a video simulation of a task, a three-dimension paper and cardboard mockup of a whole workstation, or a simple stack of hyper-linked screen shots, among other things.
In fact, a prototype can be anything from a paper-based storyboard through to a complex piece of software, and from a cardboard mockup to a molded or pressed piece of metal. A prototype allows stakeholders to interact with an envisioned product, to gain some experience of using it in a realistic setting, and to explore imagined uses.

For example, when the idea for the PalmPilot was being developed, Jeff Hawkin (founder of the company) carved up a piece of wood about the size and shape of the device he had imagined. He used to carry this piece of wood around with him and pretend to enter information into it, just to see what it would be like to own such a device (Bergman and Haitani, 2000). This is an example of a very simple (some might even say bizarre) prototype, but it served its purpose of simulating scenarios of use.

Ehn and Kyng (1991) report on the use of a cardboard box with the label "Desktop Laser Printer" as a mockup. It did not matter that, in their setup, the printer was not real. The important point was that the intended users, journalists and typographers, could experience and envision what it would be like to have one of these machines on their desks. This may seem a little extreme, but in 1982 when this was done, desktop laser printers were expensive items of equipment and were not a common sight around the office.

So a prototype is a limited representation of a design that allows users to interact with it and to explore its suitability.

© Copyright Virtual University of Pakistan

Why prototype?

Prototypes are a useful aid when discussing ideas with stakeholders; they are a communication device among team members, and are an effective way to test out ideas for yourself. The activity of building prototypes encourages reflection in design, as described by Schon (1983) and as recognized by designers from many disciplines as an important aspect of the design process. Liddle (1996), talking about software design, recommends that prototyping should always precede any writing of code.

Prototypes answer questions and support designers in choosing between alternatives. Hence, they serve a variety of purposes: for example, to test out the technical feasibility of an idea, to clarify some vague requirements, to do some user testing and evaluation, or to check that a certain design direction is compatible with the rest of the system development. Which of these is your purpose will influence the kind of prototype you build. So, for example, if you are trying to clarify how users might perform a set of tasks and whether your proposed device would support them in this, you might produce a paper-based mockup.

Low-fidelity prototyping

A low-fidelity prototype is one that does not look very much like the final product. For example, it uses materials that are very different from the intended final version, such as paper and cardboard rather than electronic screens and metal. Low-fidelity prototypes are useful because they tend to be simple, cheap, and quick to produce. This also means that they are simple, cheap, and quick to modify so they support the exploration of alternative designs and ideas. This is particularly important in early stages of development, during conceptual design for example, because prototypes that are used for exploring ideas should be flexible and encourage rather than discourage exploration and modification. Low-fidelity prototypes are never intended to be kept and integrated into the final product. They are for exploration only.

Storyboarding

Storyboarding is one example of low-fidelity prototyping that is often used in conjunction with scenarios. A storyboard consists of a series of sketches showing how a user might progress through a task using the device being developed, it can be a series of sketched screens for a GUI-based software system, or a series of scene sketches showing how a user can perform a task using the device. When used in conjunction with a scenario, the storyboard brings more detail to the written scenario and offers stakeholders a chance to role-play with the prototype, interacting with it by stepping through the scenario.

Sketching

Low-fidelity prototyping often relies on sketching, and many people find it difficult to engage in this activity because they are inhibited about the quality of their drawing. Verplank (1989) suggests that you can teach yourself to get over this inhibition. He suggests that you should devise your own symbols and icons for elements you might want to sketch, and practice using them. They don't have to be anything more than simple boxes, stick figures, and stars. Elements you might require in a storyboard sketch, for example, include "things" such as people, parts of a computer, desks, books, etc., and actions such as give, find, transfer, and write. If you are sketching an interface design, then you might need to draw various icons, dialog boxes, and so on.

High-fidelity prototyping

© Copyright Virtual University of Pakistan

High-fidelity prototyping uses materials that you would expect to be in the final product and produces a prototype that looks much more like the final thing. For example, a prototype of a software system developed in Visual Basic is higher fidelity than a paper-based mockup; a molded piece of plastic with a dummy keyboard is a higher-fidelity prototype of the PalmPilot than the lump of wood.

If you are to build a prototype in software, then clearly you need a software tool to support this. Common prototyping tools include Macromedia Director, Visual Basic, and Smalltalk. These are also full-fledged development environments, so they are powerful tools, but building prototypes using them can also be very straightforward.

Marc Rettig (1994) argues that more projects should use low-fidelity prototyj ing because of the inherent problems with high-fidelity prototyping. He identifies these problems as:

- They take too long to build.
- Reviewers and testers tend to comment on superficial aspects rather to content.
- Developers are reluctant to change something they have crafted for hours.
- A software prototype can set expectations too high.
- Just one bug in a high-fidelity prototype can bring the testing to a halt.

Type	Advantages	Disadvantages
Low-fidelity prototype	• Lower development cost. • Evaluate multiple design concepts. • Useful communication device. • Address screen layout issues. • Useful for identifying market requirements. • Proof-of-concept.	• Limited error checking. • Poor detailed specification to code to. • Facilitator-driven. • Limited utility after requirements established. • Limited usefulness for usability tests. • Navigational and flow limitations.
High-fidelity prototype	• Complete functionality. • Fully interactive. • User-driven. • Clearly defines navigational scheme. • Use for exploration and test. • Look and feel of final product. • Serves as a living specification. • Marketing and sales tool.	• M o re expensive to develop. • Time-consuming to create. • Inefficient for proof-of-concept designs. • Not effective for requirements gathering.

© Copyright Virtual University of Pakistan

Lecture 25.

Design Synthesis

Learning Goals

The aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the design principles
- Discuss the design patterns and design imperatives

In the previous lectures, we've discussed a process through which we can achieve superior interaction design. But what makes a design superior? Design that meets the goals and needs of users (without sacrificing business goals or ignoring technical constraints) is one measure of design superiority. But what are the *attributes* of a design that enable it to accomplish this successfully? Are there general, context-specific attributes and features that a design can possess to make it a "good" design?

It is strongly believed that the answer to these questions lies in the use of interaction design principles — guidelines for design of useful and useable form and behavior, and also in the use of interaction design patterns — exemplary, generalizable solutions to specific classes of design problem. This lecture defines these ideas in more detail. In addition to design-focused principles and patterns, we must also consider some larger design imperatives to set the stage for the design process.

25.1 Interaction Design Principles

Interaction design principles are generally applicable guidelines that address issues of behavior, form, and content. They represent characteristics of product behavior that help users better accomplish their goals and feel competent and confident while doing so. Principles are applied throughout the design process, helping us to translate tasks that arise out of scenario iterations into formalized structures and behaviors in the interface.

Principles minimize work

One of the primary purposes principles serve is to optimize the experience of the user when he engages with the system. In the case of productivity tools and other non-entertainment-oriented products, this optimization of experience means the *minimization of work* (Goodwin, 2002a). Kinds of work to be minimized include:

- **Logical work** — comprehension of text and organizational structures
- **Perceptual work** — decoding visual layouts and semantics of shape, size, color, and representation
- **Mnemonic work** — recall of passwords, command vectors**,** names and locations of data objects and controls, and other relationships between objects
- **Physical/motor work** — number of keystrokes, degree of mouse movement, use of gestures (click, drag, double-click), switching between input modes, extent of required navigation

© Copyright Virtual University of Pakistan

Most of the principles discussed, attempt to minimize work while providing greater levels of feedback and contextually useful information up front to the user.

Principles operate at different levels of detail

Design principles operate at three levels of organization: the conceptual level, the interaction level, and the interface level. Our focus is on interaction-level principles.

- Conceptual-level principles help define *what a product is* and how it fits into the broad context of use required by its primary personas.
- Interaction-level principles help define *how a product should behave*, in general, and in specific situations.
- Interface-level principles help define *the look and feel of interfaces*

Most interaction design principles are cross-platform, although some platforms, such as the Web and embedded systems, have special considerations based on the extra constraints imposed by that platform.

Principles versus style guides

Style guides rather rigidly define the look and feel of an interface according to corporate branding and usability guidelines. They typically focus at the detailed widget level: How many tabs are in a dialog? What should button high light states look like? What is the pixel spacing between a control and its label? These are all questions that must be answered to create a finely tuned look and feel for a product, but they don't say much about the bigger issues of what a product should be or how it should behave.
Experts recommend that designers pay attention to style guides when they are available and when fine-tuning interaction details, but there are many bigger and more interesting issues in the design of behavior that rarely find their way into style guides.
Some design principles are stated below:

Design Principles (Norman)

We have studied all of these principles in greater detail earlier, these are:

- Visibility
- Affordance
- Constraints
- Mapping
- Consistency
- Feedback

Nielsen's design principles:

Visibility of system status

Always keep users informed about what is going on, through appropriate feedback within reasonable time. For example, if a system operation will take some time, give an indication of how long and how much is complete.

© Copyright Virtual University of Pakistan

Match between system and real world

The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in natural and logical order.

User freedom and control

Users often choose system functions by mistake and need a clearly marked 'emergency exit' to leave the unwanted state without having to go through an extended dialog. Support undo and redo.

Consistency and standards

Users should not have to wonder whether words, situations or actions mean the same thing in different contexts. Follow platform conventions and accepted standards.

Error prevention

Make it difficult to make errors. Even better than good error messages is a careful design that prevents a problem from occurring in the first place.

Recognition rather than recall

Make objects, actions and options visible. The user should not have to remember information from one part of the dialog to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use

Allow users to tailor frequent actions. Accelerators - unseen by the novice user - may often speed up the interaction for the expert user to such an extent that the system can cater to both inexperienced and experienced users.

Aesthetic and minimalist design

Dialogs should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialog competes with the relevant units of information and diminishes their relative.

Help users recognize, diagnose, and recover from errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and documentation

Few systems can be used with no instruction so it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete step to be carried out, and not be too large.

© Copyright Virtual University of Pakistan

Design Principles (Simpson, 1985)

- Define the users
- Anticipate the environment in which your program will be used
- Give the operators control
- Minimize operators' work
- Keep the program simple
- Be consistent
- Give adequate feedback
- Do not overstress working memory
- Minimize dependence on recall memory
- Help the operators remain oriented
- Code information properly (or not at all)
- Follow prevailing design conventions

Design Principles (Shneiderman, 1992)

1. *Strive for consistency* in action sequences, layout, terminology, command use and so on.
2. *Enable frequent users to use shortcuts*, such as abbreviations, special key sequences and macros, to perform regular, familiar actions more quickly.
3. *Offer informative feedback* for every user action, at a level appropriate to the magnitude of the action.
4. *Design dialogs to yield closure* so that the user knows when they have completed a task.
5. *Offer error prevention and simple error handling* so that, ideally, users are prevented from making mistakes and, if they do, they are offered clear and informative instructions to enable them to recover,
6. *Permit easy reversal of actions* in order to relieve anxiety and encourage exploration, since the user knows that he can always return to the previous state.
7. *Support internal locus of control* so that the user is in control of the system, which responds to his actions.
8. *Reduce short-term memory load* by keeping displays simple, consolidating multiple page displays and providing time for learning action sequences.

These rules provide useful shorthand for the more detailed sets of principles described earlier. Like those principles, they are not applicable to every eventuality and need to be interpreted for each new situation. However, they are broadly useful and their application will only help most design projects.

Design Principles (Dumas, 1988)

- Put the user in control
- Address the user's level of skill and knowledge
- Be consistent in wording, formats, and procedures
- Protect the user from the inner workings of the hardware and software that is behind the interface

© Copyright Virtual University of Pakistan

- Provide online documentation to help the user to understand how to operate the application and recover from errors
- Minimize the burden on user's memory
- Follow principles of good graphics design in the layout of the information on the screen

25.2 Interaction Design Patterns

Design patterns serve two important functions. The first function is to capture useful design decisions and generalize them to address similar classes of problems in the future (Borchers, 2001). In this sense, patterns represent both the capture and formalization of design knowledge, which can serve many purposes. These include reducing design time and effort on new projects, educating designers new to a project, or — if the pattern is sufficiently broad in its application — educating designers new to the field.

Although the application of patterns in design pedagogy and efficiency is certainly important, the key factor that makes patterns exciting is that they can represent optimal or near-optimal interactions for the user and the class of activity that the pattern addresses.

Interaction and architectural patterns

Interaction design patterns are far more akin to the architectural design patterns first envisioned by Christopher Alexander in his seminal volumes *A Pattern Language* (1977) and *The Timeless Way of Building* (1979) than they are to the popular engineering use of patterns. Alexander sought to capture in a set of building blocks something that he called "the quality without a name," that essence of architectural design that creates a feeling of well-being in the inhabitants of architectural structures. It is this human element that differentiates interaction design patterns (and architectural design patterns) from engineering design patterns, whose sole concern is efficient reuse of code.

One singular and important way that interaction design patterns differ from architectural design patterns is their concern, not only with structure and organization of elements, but with dynamic behaviors and changes in elements in response to user activity. It is tempting to view the distinction simply as one of change over time, but these changes are interesting because they occur in response to human activity. This differentiates them from preordained temporal transitions that can be found in artifacts of broadcast and film media (which have their own distinct set of design patterns). Jan Borchers (2001) aptly describes interaction design patterns:

> [Interaction design] Patterns refer to relationships between physical elements and the events that happen there. Interface designers, like urban architects, strive to create environments that establish certain behavioral patterns with a positive effect on those people 'inside' these environments . . . 'timeless' architecture is comparable to user interface qualities such as 'transparent' and 'natural.'

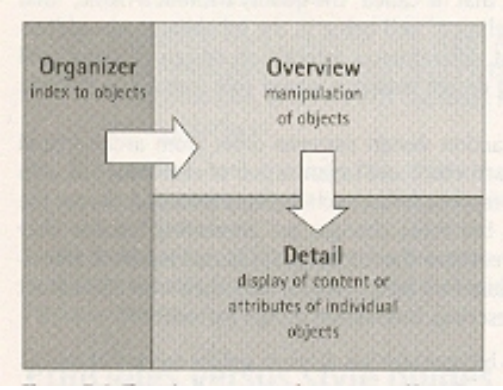
Types of interaction design patterns

Like most other design patterns, interaction design patterns can be hierarchically organized from the system level down to the level of individual interface widgets. Like principles, they can be applied at different levels of organization (Goodwin, 2002a):

© Copyright Virtual University of Pakistan

- Postural patterns can be applied at the conceptual level and help determine the overall product stance in relation to the user.
- Structural patterns solve problems that relate to the management of information display and access, and to the way containers of data and functions are visually manipulated to best suit user goals and contexts. They consist of views, panes, and element groupings.
- Behavioral patterns solve wide-ranging problems relating to specific interactions with individual functional or data objects or groups of such objects. What most people think of as system and widget behaviors fall into this category.

Structural patterns are perhaps the least-documented patterns, but they are nonetheless in widespread use. One of the most commonly used high-level structural patterns is apparent in Microsoft Outlook with its navigational pane on the left, overview pane on the upper right, and detail pane on the lower right (see Figure).



This pattern is optimal for full-screen applications that require user access to many different kinds of objects, manipulation of those objects in groups, and display of detailed content or attributes of individual objects or documents. The pattern permits all this to be done smoothly in a single screen without the need for additional windows. Many e-mail clients make use of this pattern, and variations of it appear in many authoring and information management tools where rapid access to and manipulation of many types of objects is common

Structural patterns, pattern nesting, and pre-fab design

Structural patterns often contain other structural patterns; you might imagine that a comprehensive catalogue of structural patterns could, given a clear idea of user needs, permit designers to assemble coherent, Goal-Directed designs fairly rapidly. Although there is some truth in this assertion, which the experts have observed in practice, it is simply never the case that patterns can be mechanically assembled in cookie-cutter fashion. As Christopher Alexander is swift to point out (1979), architectural patterns are the antithesis of the pre-fab building, because context is of absolute importance in defining the actual rendered form of the pattern in the world. The environment where the pattern is deployed is critical, as are the other patterns that comprise it, contain it, and abut it. The same is true for interaction design patterns. The core of each pattern lies in the relationships between represented objects and between those objects and the goals of the user. The precise form of the pattern is certain to be somewhat different for each

© Copyright Virtual University of Pakistan

instance, and the objects that define it will naturally vary from domain to domain. But the relationships between objects remain essentially the same.

25.3 Interaction Design Imperatives

Beyond the need for principles of the type described previously, the experts also feel a need for some even more fundamental principles for guiding the design process as a whole. The following set of top-level design imperatives (developed by Robert Reimann, Hugh Dubberly, Kim Goodwin, David Fore, and Jonathan Korman) apply to interaction design, but could almost equally well apply to any design discipline. Interaction designers should create design solutions that are

- Ethical *[considerate, helpful]*

  Do no harm
  Improve human situations

- Purposeful *[useful, usable]*

  Help users achieve their goals and aspirations
  Accommodate user contexts and capacities

- Pragmatic *[viable, feasible*

  Help commissioning organizations achieve their goals 7
  Accommodate business and technical requirements

- Elegant *[efficient, artful, affective]*

  Represent the simplest complete solution
  Possess internal (self-revealing, understandable) coherence
  Appropriately accommodate and stimulate cognition and emotion

- Ask relevant questions when planning manuals
- Learn about your audiences
- Understand how people use manuals
- Organize so that users can find information quickly
- Put the user in control by showing the structure of the manual
- Use typography to give readers clues to the structure of the manual
- Write so that users can picture themselves in the text
- Write so that you don’t overtax users’ working memory
- Use users’ words
- Be consistent
- Test for usability
- Expect to revise
- Understand who uses the product and why
- Adapt the dialog to the user
- Make the information accessible
- Apply a consistent organizational strategy

© Copyright Virtual University of Pakistan

- Make messages helpful
- Prompt for inputs
- Report status clearly
- Explain errors fully
- Fir help smoothly into the users' workflow

© Copyright Virtual University of Pakistan

Lecture 26.

Behavior & Form Part I

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the narratives and scenarios
- Define requirements using persona-based design

26.1 Software Posture

Most people have a predominant behavioral stance that fits their working role on the job: The soldier is wary and alert; the toll-collector is bored and disinterested; the actor is flamboyant and bigger than life; the service representative is upbeat and helpful. Programs, too, have a predominant manner of presenting themselves to the user.

A program may be bold or timid, colorful or drab, but it should be so for a specific, goal-directed reason. Its manner shouldn't result from the personal preference of its designer or programmer. The presentation of the program affects the way the user relates to it, and this relationship strongly influences the usability of the product. Programs whose appearance and behavior conflict with their purposes will seem jarring and inappropriate, like fur in a teacup or a clown at a wedding.

The look and behavior of your program should reflect how it is used, rather than an arbitrary standard. A program's behavioral stance — the way it presents itself to the user — is its posture. The look and feel of your program from the perspective of posture is *not* an aesthetic choice: It is a behavioral choice. Your program's posture is its behavioral foundation, and whatever aesthetic choices you make should be in harmony with this posture.

The posture of your interface tells you much about its behavioral stance, which, in turn, dictates many of the important guidelines for the rest of the design. As an interaction designer, one of your first design concerns should be ensuring that your interface presents the posture that is most appropriate for its behavior and that of your users. This lecture explores the different postures for applications on the desktop.

26.2 Postures for the Desktop

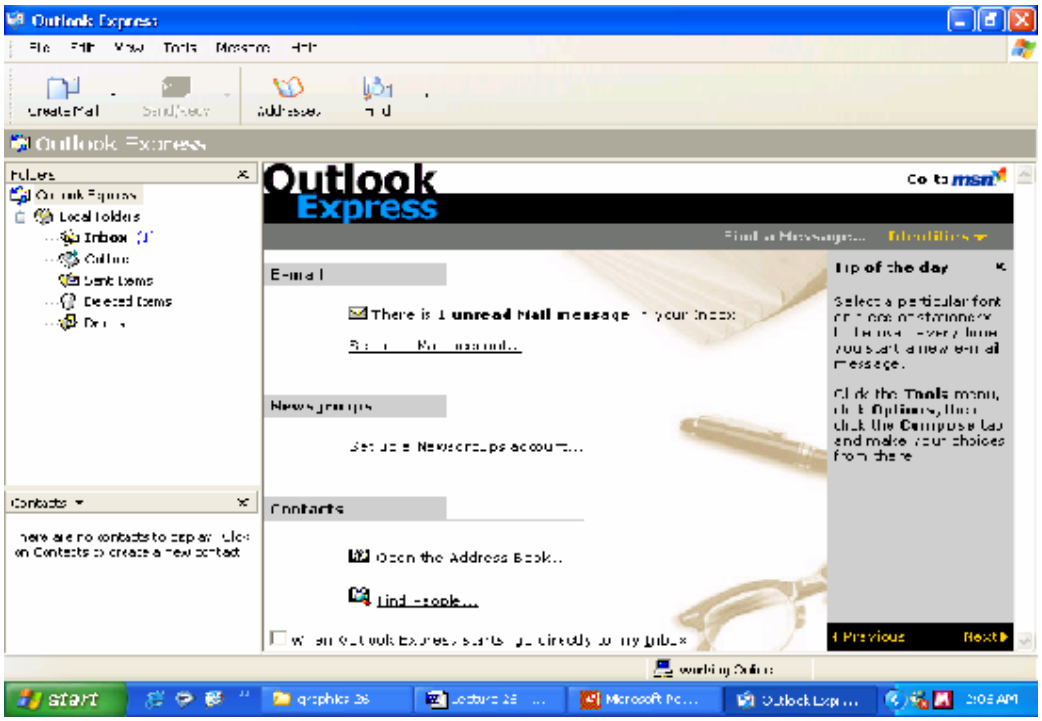
Desktop applications fit into four categories of posture: sovereign, transient, daemonic, and auxiliary. Because each describes a different set of behavioral attributes, each also describes a different type of user interaction. More importantly, these categories give the designer a point of departure for designing an interface. A sovereign posture program, for example, won't feel right unless it behaves in a "sovereign" way. Web and other non-desktop applications have their own variations of posture.

Sovereign posture

Programs that are best used full-screen, monopolizing the user's attention for long periods of time, are sovereign posture application. Sovereign applications offer a large set of

© Copyright Virtual University of Pakistan

related functions and features, and users tend to keep them up and running continuously. Good examples of this type of application are word processors, spreadsheets, and e-mail applications. Many vertical applications are also sovereign applications because they often deploy on the screen for long periods of time, and interaction with them can be very complex and involved. Users working with sovereign programs often find themselves in a state of flow. Sovereign programs are usually used maximized. For example, it is hard to imagine using Outlook in a 3x4 inch window — at that size it's not really appropriate for its main job: creating and viewing e-mail and appointments (see Figure).



Sovereign programs are characteristically used for long, continuous stretches of time. A sovereign program dominates a user's workflow as his primary tool. PowerPoint, for example, is open full screen while you create a presentation from start to finish. Even if other programs are used for support tasks, PowerPoint maintains its sovereign stance.

The implications of sovereign behavior are subtle, but quite clear after you think about them. The most important implication is that users of sovereign programs are intermediate users. Each user spends time as a novice, but only a short period of time *relative to the amount of time he will eventually spend* using the product. Certainly a new user has to get over the painful hump of an initial learning curve, but seen from the perspective of the entire relationship of the user with the application, the time he spends getting acquainted with the program is small.

From the designer's point of view, this means that the program should be designed for optimal use by perpetual intermediates and not be aimed primarily for beginners (or experts). Sacrificing speed and power in favor of a clumsier but easier-to-learn idiom is out of place here, as is providing only nerdy power tools. Of course, if you can offer easier idioms without compromising the interaction for intermediate users; that is always

© Copyright Virtual University of Pakistan

best. Between first-time users and intermediate users there are many people who use sovereign applications only on occasion. These infrequent users cannot be ignored.

However, the success of a sovereign application is still dependent on its intermediate, frequent users until someone else satisfies both them *and* inexperienced users. WordStar, an early word processing program, is a good example. It dominated the word processing marketplace in the late 70s and early 80s because it served its intermediate users exceedingly well, even though it was extremely difficult for infrequent and first-time users. WordStar Corporation thrived until its competition offered the same power for intermediate users while simultaneously making it much less painful for infrequent users. WordStar, unable to keep up with the competition, rapidly dwindled to insignificance.

TAKE THE PIXELS

Because the user's interaction with a sovereign program dominates his session at the computer, the program shouldn't be afraid to take as much screen real estate as possible. No other program will be competing with yours, so expect to take advantage of it all. Don't waste space, but don't be shy about taking what you need to do the job. If you need four toolbars to cover the bases, use four toolbars. In a program of a different posture, four toolbars may be overly complex, but the sovereign posture has a defensible claim on the pixels.

In most instances, sovereign programs run maximized. In the absence of explicit instructions from the user, your sovereign application should default to maximized (full-screen) presentation. The program needs to be fully resizable and must work reasonably well in other screen configurations, but it must optimize its interface for full-screen instead of the less likely cases.

Because the user will stare at a sovereign application for long periods, you should take care to mute the colors and texture of the visual presentation. Keep the color palette narrow and conservative. Big colorful controls may look really cool to newcomers, but they seem garish after a couple of weeks of daily use. Tiny dots or accents of color will have more effect in the long run than big splashes, and they enable you to pack controls together more tightly than you could otherwise. Your user will stare at the same palettes, menus, and toolbars for many hours, gaining an innate sense of where things are from sheer familiarity. This gives you, the designer, and freedom to do more with fewer pixels. Toolbars and their controls can be smaller than normal. Auxiliary controls like screen-splitters, rulers, and scroll bars can be smaller and more closely spaced.

RICH VISUAL FEEDBACK

Sovereign applications are great platforms for creating an environment rich in visual feedback for the user. You can productively add extra little bits of information into the interface. The status bar at the bottom of the screen, the ends of the space normally occupied by scroll bars, the title bar, and other dusty corners of the program's visible extents can be filled with visual indications of the program's status, the status of the data, the state of the system, and hints for more productive user actions. However, be careful: While enriching the visual feedback, you must be careful not to create an interface that is hopelessly cluttered.

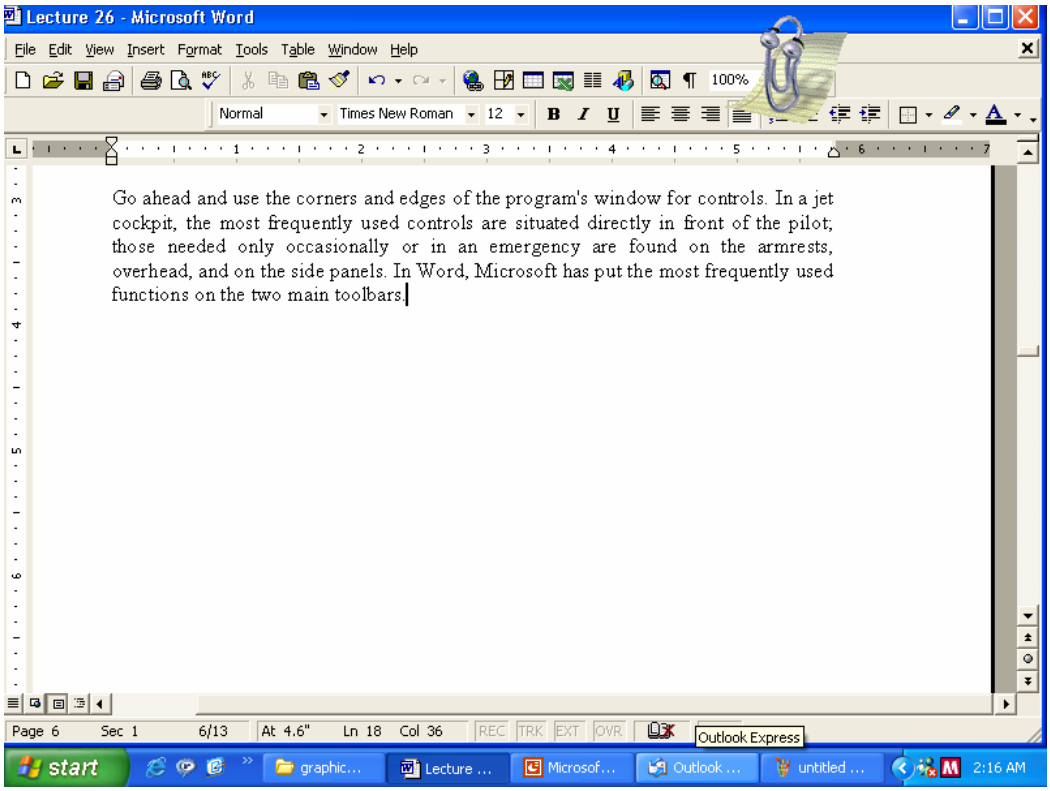
© Copyright Virtual University of Pakistan

The first-time user won't even notice such artifacts, let alone understand them, because of the subtle way they are shown on the screen. After a couple of months of steady use, however, he will begin to see them, wonder about their meaning, and experimentally explore them. At this point, the user will be willing to expend a little effort to learn more. If you provide an easy means for him to find out what the artifacts are, he will become not only a better user, but a more satisfied user, as his power over the program grows with his understanding. Adding such richness to the interface is like adding a variety of ingredients to a meat stock — it enhances the entire meal.

RICH INPUT

Sovereign programs similarly benefit from rich input. Every frequently used aspect of the program should be controllable in several ways. Direct manipulation, dialog boxes, keyboard mnemonics, and keyboard accelerators are all appropriate. You can make more aggressive demands on the user's fine motor skills with direct-manipulation idioms. Sensitive areas on the screen can be just a couple of pixels across because you can assume that the user is established comfortably in his chair, arm positioned in a stable way on his desk, rolling his mouse firmly across a resilient mouse pad.

Go ahead and use the corners and edges of the program's window for controls. In a jet cockpit, the most frequently used controls are situated directly in front of the pilot; those needed only occasionally or in an emergency are found on the armrests, overhead, and on the side panels. In Word, Microsoft has put the most frequently used functions on the two main toolbars.



They put the frequently used but visually dislocating functions on small controls to the left of the horizontal scroll bar near the bottom of the screen. These controls change the

© Copyright Virtual University of Pakistan

appearance of the entire visual display — Normal view, Page layout view and Outline view. Neophytes do not often use them and, if accidentally triggered, they can be confusing. By placing them near the bottom of the screen, they become almost invisible to the new user. Their segregated positioning subtly and silently indicates that caution should be taken in their use. More experienced users, with more confidence in their understanding and control of the program, will begin to notice these controls and wonder about their purpose. They can experimentally select them when they feel fully prepared for their consequence. This is a very accurate and useful mapping of control placement to usage.

The user won't appreciate interactions that cause a delay. Like a grain of sand in your shoe, a one- or two-second delay gets painful after a few repetitions. It is perfectly acceptable for functions to take time, but they should not be frequent or repeated procedures during the normal use of the product. If, for example, it takes more than a fraction of a second to save the user s work to disk, the user quickly comes to view that delay as unreasonable. On the other hand, inverting a matrix or changing the entire formatting style of a document can take a few seconds without causing irritation because the user can plainly see what a big job it is. Besides, he won't invoke it very often.

DOCUMENT-CENTRIC APPLICATIONS

The dictum that sovereign programs should fill the screen is also true of document windows within the program itself. Child windows containing documents should always be maximized inside the program unless the user explicitly instructs otherwise.

Many sovereign programs are also document-centric (their primary functions involve the creation and viewing of documents containing rich data), making it easy to confuse the two, but they are not the same. Most of the documents we work with are 8½-by-l1 inches and won't fit on a standard computer screen. We strain to show as much of them as possible, which naturally demands a full-screen stance. If the document under construction were a 32x32 pixel icon, for example, a document-centric program wouldn't need to take the full screen. The sovereignty of a program does not come from its document-centricity nor from the size of the document — it comes from the nature of the program's use.

If a program manipulates a document but only performs some very simple, single function, like scanning in a graphic, it isn't a sovereign application and shouldn't exhibit sovereign behavior. Such single-function applications have a posture of their own, the transient posture.

Transient posture

A transient posture program comes and goes, presenting a single, high-relief function with a tightly restricted set of accompanying controls. The program is called when needed, appears, performs its job, and then quickly leaves, letting the user continue his more normal activity, usually with a sovereign application.

The salient characteristic of transient programs is their temporary nature. Because they don't stay on the screen for extended periods of time, the user doesn't get the chance to become very familiar with them. Consequently, the program’s user interface needs to be unsubtle, presenting its controls clearly and boldly with no possibility of mistakes. The

© Copyright Virtual University of Pakistan

interface must spell out what it does: This is not the place for artistic-but-ambiguous images or icons — it *is* the place for big buttons with precise legends spelled out in a slightly oversized, easy-to-read typeface.

Although a transient program can certainly operate alone on your desktop, it usually acts in a supporting role to a sovereign application. For example, calling up the Explorer to locate and open a file while editing another with Word is a typical transient scenario. So is setting your speaker volume. Because the transient program borrows space at the expense of the sovereign, it must respect the sovereign by not taking more space on screen than is absolutely necessary. Where the sovereign can dig a hole and pour a concrete foundation for itself, the transient program is just on a weekend campout. It cannot deploy itself on screen either graphically or temporally. It is the taxicab of the software world.

BRIGHT AND CLEAR

Whereas a transient program must consent the total amount of screen real estate it consumes, the controls on its surface can be proportionally larger than those on a sovereign application. Where such heavy-handed visual design on a sovereign program would pall within a few weeks, the transient program isn't on screen long enough for it to bother the user. On the contrary, the bolder graphics help the user to orient him more quickly when the program pops up. The program shouldn't restrict itself to a drab palette, but should instead paint itself in brighter colors to help differentiate it from the hosting sovereign, which will be more appropriately shaded in muted hues. Transient programs should use their brighter colors and bold graphics to clearly convey their purpose — the user needs big, bright, reflective road signs to keep him from making the wrong turn at 100 kilometers per hour.

Transient programs should have instructions built into their surface. The user may only see the program once a month and will likely forget the meanings of the choices presented. Instead of a button captioned Setup, it might be better to make the button large enough to caption it Setup User Preferences. The meaning is clearer, and the button more reassuring. Likewise, nothing should be abbreviated on a transient program —everything should be spelled out to avoid confusion. The user should be able to see without difficulty that the printer is busy, for example, or that the audio is five seconds long.

KEEP IT SIMPLE

After the user summons a transient program, all the information and facilities he needs should be right there on the surface of the program's single window. Keep the user's focus of attention on that window and never force him into supporting subwindows or dialog boxes to take care of the main function of the program. If you find yourself adding a dialog box or second view to a transient application, that's a key sign that your design needs a review.

Transient programs are not the place for tiny scroll bars and fussy point-click-and-drag interfaces. You want to keep the demands here on the user's fine motor skills down to a minimum. Simple push-buttons for simple functions are better. Anything directly manipulability must be big enough to move to easily: at least twenty pixels square. Keep controls off the borders of the window. Don't use the window bottoms, status bars, or sides in transient programs. Instead, position the controls up close and personal in the

© Copyright Virtual University of Pakistan

main part of the window. You should definitely provide a keyboard interface, but it must be a simple one. It shouldn't be more complex than Enter, Escape, and Tab. You might add the arrow keys, too, but that's about it.

Of course, there are exceptions to the monothematic nature of transient programs, although they are rare. If a transient program performs more than just a single function, the interface should communicate this visually. For example, if the program imports and exports graphics, the interface should be evenly and visually split into two halves by bold coloration or other graphics One half could contain the controls for importing and the other half the controls for exporting The two halves must be labeled unambiguously. Whatever you do, don't add more windows or dialogs.

Keep in mind that any given transient program may be called upon to assist in the management of some aspect of a sovereign program. This means that the transient program, as it positions itself on top of the sovereign, may obscure the very information that it is chartered to work on. This implies that the transient program must be movable, which means it must have a title bar.

It is vital to keep the amount of management' overhead as low as possible with transient programs. All the user wants to do is call the program up, request a function, and then end the program. It is completely unreasonable to force the user to add non-productive window-management tasks to this interaction.

REMEMBERING STATE

The most appropriate way to help the user with both transient and sovereign apps is to give the program a memory. If the transient program remembers where it was the last time it was used, the chances are excellent that the same size and placement will be appropriate next time, too. It will almost always be more apt than any default setting might chance to be. Whatever shape and position the user morphed the program into is the shape and position the program should reappear in when it is next summoned. Of course, this holds true for its logical settings, too.

On the other hand, if the use of the program is really simple and single-minded, go ahead and specify its shape — omit the frame, the directly resizable window border. Save yourself the work and remove the complexity from the program (be careful, though, as this can certainly be abused). The goal here is not to save the programmer work — that's just a collateral benefit — but to keep the user aware of as few complexities as possible. If the program's functions don't demand resizing and the overall size of the program is small, the principle that simpler is better takes on more importance than usual. The calculator accessory' in Windows and on the Mac, for example, isn't resizable. It is always the correct size and shape.

No doubt you have already realized that almost all dialog boxes are really transient programs. You can see that all the preceding guidelines for transient programs apply equally well to the design of dialog boxes.

© Copyright Virtual University of Pakistan

Daemonic posture

Programs that do not normally interact with the user are daemonic posture programs. These programs serve quietly and invisibly in the background, performing possibly vital tasks without the need for human intervention. A printer driver is an excellent example.
As you might expect, any discussion of the user interface of daemonic programs is necessarily j short. Too frequently, though, programmers give daemonic programs full-screen control panels that are better suited to sovereign programs. Designing your fax manager in the image of Excel, for example, is a fatal mistake. At the other end of the spectrum, daemonic programs are, too frequently, unreachable by the user, causing no end of frustration when adjustments need to be made.

Where a transient program controls the execution of a function, daemonic programs usually manage processes. Your heartbeat isn't a function that must be consciously controlled; rather, it is a process that proceeds autonomously in the background. Like the processes that regulate your heartbeat, daemonic programs generally remain completely invisible, competently performing their process as long as your computer is turned on. Unlike your heart, however, daemonic programs must occasionally be installed and removed and, also occasionally, they must be adjusted to deal with changing circumstances. It is at these times that the daemon talks to the user. Without exception, the interaction between the user and a daemonic program is transient in nature, and all the imperatives of transient program design hold true here also.

The principles of transient design that are concerned with keeping the user informed of the purpose of the program and of the scope and meaning of the user's available choices become even more critical with daemonic programs. In many cases, the user will not even be consciously (or unconsciously) aware of the existence of the daemonic program. If you recognize that, it becomes obvious that reports about status from that program can be quite dislocating if not presented in an appropriate context. Because many of these programs perform esoteric functions — like printer drivers or communications concentrators — the messages from them must take particular care not to confuse the user or lead to misunderstandings.

A question that is often taken for granted with programs of other postures becomes very significant with daemonic programs: If the program is normally invisible, how should the user interface be summoned on those rare occasions when it is needed? One of the most frequently used methods is to represent the daemon with an on-screen program icon found either in the status area (system tray) in Windows or in the far right of the Mac OS menu bar. Putting the icon so boldly in the user's face when it is almost never needed is a real affront, like pasting an advertisement on the windshield of somebody's car. If your daemon needs configuring no more than once a day, get it off of the main screen. Windows XP now hides daemonic icons that are not actively being used. Daemonic icons should only be employed permanently if they provide continuous, useful status information.

Microsoft makes a bit of a compromise here by setting aside an area on the far-right side of the taskbar as a status area wherein icons belonging to daemonic posture programs may reside. This area, also known as the system tray, has been abused by programmers, who often use it as a quick launch area for sovereign applications. As of Windows XP, Microsoft set the standard that only status icons are to appear in the status area (a quick launch area is supported next to the Start button on the taskbar), and unless the user

© Copyright Virtual University of Pakistan

chooses otherwise, only icons actively reporting status changes will be displayed. Any others will be hidden. These decisions are very appropriate handling of transient programs.

An effective approach for configuring daemonic programs is employed by both the Mac and Windows: control panels, which are transient programs that run as launch able applications to configure daemons. These give the user a consistent place to go for access to such process-centric applications.

Auxiliary posture

Programs that blend the characteristics of sovereign and transient programs exhibit auxiliary posture. The auxiliary program is continuously present like a sovereign, but it performs only a supporting role. It is small and is usually superimposed on another application the way a transient is. The Windows taskbar, clock programs, performance monitors on many Unix platforms, and Stickies on the Mac are all good examples of auxiliary programs. People who continuously use instant messaging applications are also using them in an auxiliary manner. In Windows XP's version of Internet Explorer, Microsoft has recognized the auxiliary role that streaming audio can play while the user is browsing the Web. It has integrated its audio player into a side pane in the browser.

Auxiliary programs are typically silent reporters of ongoing processes, although some, like Stickies or stock tickers, are for displaying other data the user is interested in. In some cases, this reporting may be a function that they perform in addition to actually managing processes, but this is not necessarily true. An auxiliary application may, for example, monitor the amount of system resources either in use or available. The program constantly displays a small bar chart reflecting the current resource availability.

A process-reporting auxiliary program must be simple and often bold in reporting its information. It must be very respectful of the pre-eminence of sovereign programs and should be quick to move out of the way when necessary.

Auxiliary programs are not the locus of the user's attention; that distinction belongs to the host application. For example, take an automatic call distribution (ACD) program. An ACD is used to evenly distribute incoming calls to teams of customer-service representatives trained either to take orders, provide support, or both. Each representative uses a computer running an application specific to his or her job. This application, the primary reason for the system's purchase, is a sovereign posture application; the ACD program is an auxiliary application on top of it. For example, a sales agent fields calls from prospective buyers on an incoming toll-free number. The representative's order entry program is the sovereign, whereas the ACD program is the auxiliary application, riding on top to feed incoming calls to the agent. The ACD program must be very conservative in its use of pixels because it always obscures some of the underlying sovereign application. It can afford to have small features because it is on the screen for long periods of time. In other words, the controls on the auxiliary application can be designed to a sovereign's sensibilities.

© Copyright Virtual University of Pakistan

Lecture 27.

Behavior & Form Part II

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the narratives and scenarios
- Define requirements using persona-based design

27.1 Postures for the Web

Designers may be tempted to think that the Web is, by necessity, different from desktop applications in terms of posture. Although there are variations that exhibit combinations of these postures, the basic four stances really do cover the needs of most Web sites and Web.

Information-oriented sites

Sites that are purely informational, which require no complex transactions to take place beyond navigating from page to page and limited search, must balance two forces: the need to display a reasonable density of useful information, and the need to allow first time and infrequent users to easily learn and navigate the site. This implies a tension between sovereign and transient attributes in informational sites. Which stance is more dominant depends largely on who the target personas are and what their behavior patterns are when using the site: Are they infrequent or one-time users, or are they repeat users who will return weekly or daily to view content?

The frequency at which content can be updated on a site does, in some respects, influence this behavior: Informational sites with daily-updated information will naturally attract repeat users more than a monthly-updated site. Infrequently updated sites may be used more as occasional reference (assuming the information is not too topical) rather than heavy repeat use and should then be given more of a transient stance than a sovereign one. What's more, the site can configure itself into a more sovereign posture by paying attention to how often that particular user visits.

SOVEREIGN ATTRIBUTES

Detailed information display is best accomplished by assuming a sovereign stance. By assuming full-screen use, designers can take advantage of all the possible space available to clearly present both the information itself and the navigational tools and cues to keep users oriented.

The only fly in the ointment of sovereign stance on the Web is choosing which full-screen resolution is appropriate. In fact, this is an issue for desktop applications as well. The only difference between the desktop and the Web in this regard is that Web sites have little leverage in influencing what screen resolution users will have. Users, however, who are spending money on expensive desktop productivity applications, will probably make sure that they have the right hardware to support the needs of the software. Thus, Web

© Copyright Virtual University of Pakistan

designers need to make a decision early on what the lowest common denominator they wish to support will be. Alternatively, they must code more complex sites that may be optimal on higher resolution screens, but which are still usable (without horizontal scrolling) on lower-resolution monitors.

TRANSIENT ATTRIBUTES

The less frequently your primary personas access the site, the more transient a stance the site needs to take. In an informational site, this manifests itself in terms of ease and clarity of navigation.

Sites used for infrequent reference might be bookmarked by users: You should make it possible for them to bookmark any page of information so that they can reliably return to it at any later time.

Users will likely visit sites with weekly to monthly updated material intermittently, and so navigation there must be particularly clear. If the site can retain information about past user actions via cookies or server-side methods and present information that is organized based on what interested them previously, this could dramatically help less frequent users find what they need with minimal navigation.

Transactional sites and Web applications

Transactional sites allow users to buy, sell, or exchange things online, while web applications let users interact and perform tasks like editing, booking, or messaging.

Transactional Web sites and Web applications have many of the same tensions between sovereign and transient stances that informational sites do. This is a particular challenge because the level of interaction can be significantly more complex.

Again, a good guide is the goals and needs of the primary personas: Are they consumers, who will use the site at their discretion, perhaps on a weekly or monthly basis, or are they employees who (for an enterprise or B2B Web application) must use the site as part of their job on a daily basis? Transactional sites that are used for a significant part of an employee's job should be considered full sovereign applications.

On the other hand, e-commerce, online banking, and other consumer-oriented transactional sites must, like informational sites, balance between sovereign and transient stances very similarly to informational sites. In fact, many consumer transactional sites have a heavy informational aspect because users like to research and compare products, investments, and other items to be transacted upon. For these types of sites, navigational clarity is very important, as is access to supporting information and the streamlining of transactions. Amazon.com has addressed many of these issues quite well, via one-click ordering, good search and browsing capability, online reviews of items, recommendation lists, persistent shopping cart, and tracking of recently viewed items. If Amazon has a fault, it may be that it tries to do a bit too much: Some of the navigational links near the bottom of the pages likely don't get hit very often.

27.2 Web portals

Early search engines allowed people to find and access content and functions distributed throughout the world on the Web. They served as portals in the original sense of the word — ways to get somewhere else. Nothing really happens in these navigational portals; you get in, you go somewhere, you get out. They are used exclusively to gain access quickly to unrelated information and functions.

© Copyright Virtual University of Pakistan

If the user requires access via a navigational portal relatively infrequently, the appropriate posture is transient, providing clear, simple navigational controls and getting out of the way. If the user needs more frequent access, the appropriate posture is auxiliary: a small and persistent panel of links (like the Windows taskbar).

As portal sites evolved, they offered more integrated content and function and grew beyond being simple stepping-stones to another place. Consumer-oriented portals provide unified access to content and functionality related to a specific topic, and enterprise portals provide internal access to important company information and business tools. In both cases, the intent is essentially to create an environment in which users can access a particular kind of information and accomplish a particular kind of work: environmental portals. Actual work is done in an environmental portal. Information is gathered from disparate sources and acted upon; various tools are brought together to accomplish a unified purpose.

An environmental portal's elements need to relate to one another in a way that helps users achieve a specific purpose. When a portal creates a working environment, it also creates a sense of place; the portal is no longer a way to get somewhere else, but a destination in and of itself. The appropriate posture for an environmental portal is thus sovereign.

Within an environmental portal, the individual elements function essentially as small applications running simultaneously — as such, the elements themselves also have postures:

- **Auxiliary elements**: Most of the elements in an environmental portal have an auxiliary posture; they typically present aggregated sets of information to which the user wants constant access (such as dynamic status monitors), or simple functionality (small applications, link lists, and so on). Auxiliary elements are, in fact, the key building block of environmental portals. The sovereign portal is, therefore, composed of a set of auxiliary posture mini-applications.
- **Transient elements**: In addition to auxiliary elements, an environmental portal often provides transient portal services as well. Their complexity is minimal; they are rich in explanatory elements, and they are used for short periods on demand. Designers should give a transient posture to any embedded portal service that is briefly and temporarily accessed (such as a to-do list or package-tracking status display) so that it does not compete with the sovereign/auxiliary posture of the portal itself. Rather it becomes a natural, temporary extension of it.

When migrating traditional applications into environmental portals, one of the key design challenges is breaking the application apart into a proper set of portal services with auxiliary or transient posture. Sovereign, full-browser applications are not appropriate within portals because they are not perceived as part of the portal once launched.

27.3 Postures for Other Platforms

Handheld devices, kiosks, and software-enabled appliances each have slightly different posture issues. Like Web interfaces, these other platforms typically express a tension between several postures.

© Copyright Virtual University of Pakistan

Kiosks

The large, full-screen nature of kiosks would appear to bias them towards sovereign posture, but there are several reasons why the situation is not quite that simple. First, users of kiosks are often first-time users (with the exception, perhaps, of ATM users), and are in most cases not daily users. Second, most people do not spend any significant amount of time in front of a kiosk: They perform a simple transaction or search, get what information they need, and then move on. Third, most kiosks employ either touch screens or bezel buttons to the side of the display, and neither of these input mechanisms support the high data density you would expect of a sovereign application. Fourth, kiosk users are rarely comfortably seated in front of an optimally placed monitor, but are standing in a public place with bright ambient light and many distractions. These user behaviors and constraints should bias most kiosks towards transient posture, with simple navigation, large controls, and rich visuals to attract attention and hint at function.

Educational and entertainment kiosks vary somewhat from the strict transient posture required of more transactional kiosks. In this case, exploration of the kiosk environment is more important than the simple completion of single transactions or searches. In this case, more data density and more complex interactions and visual transitions can sometimes be introduced to positive effect, but the limitations of the input mechanisms need to be carefully respected, lest the user lose the ability to successfully navigate the interface.

Handheld devices

Designing for handheld devices is an exercise in hardware limitations: input mechanisms, screen size and resolution, and power consumption, to name a few. One of the most important insights that many designers have now realized with regard to handheld devices is that handhelds are often not standalone systems. They are, as in the case of personal information managers like Palm and Pocket PC devices, satellites of a desktop system, used more to view information than perform heavy input on their own. Although folding keyboards can be purchased for many handhelds, this, in essence, transforms them into desktop systems (with tiny screens). In the role of satellite devices, an auxiliary posture is appropriate for the most frequently used handheld applications — typical PIM, e-mail, and Web browsing applications, for example. Less frequently or more temporarily used handheld applications (like alarms) can adopt a more transient posture.

Cellular telephones are an interesting type of handheld device. Phones are not satellite devices: they are primary communication devices. However, from an interface posture standpoint, phones are really transient. You place a call as quickly as possible and then abandon the interface to your conversation. The best interface for a phone is arguably non-visual. Voice activation is perfect for placing a call; opening the flip lid on a phone is probably the most effective way of answering it (or again using voice activation for hands-free use). The more transient the phones interface is, the better.

In the last couple of years, handheld data devices and handheld phones have been converging. These convergence devices run the risk of making phone operation too complex and data manipulation too difficult, but the latest breed of devices like the Handspring Treo has delivered a successful middle ground. In some ways, they have made the phone itself more usable by allowing the satellite nature of the device to aid in the input of information to the phone: Treos make use of desktop contact information to

© Copyright Virtual University of Pakistan

synchronize the device's phonebook, for example, thus removing the previously painful data entry step and reinforcing the transient posture of the phone functionality. It is important, when designing for these devices, to recognize the auxiliary nature of data functions and the transient nature of phone functions, using each to reinforce the utility of the other. (The data dialup should be minimally transient, whereas the data browsing should be auxiliary

Appliances

Most appliances have extremely simple displays and rely heavily on hardware buttons and dials to manipulate the state of the appliance. In some cases, however, major appliances (notably washers and dryers) will sport color LCD touch screens allowing rich output and direct input.

Appliance interfaces, like the phone interfaces mentioned in the previous section, should primarily be considered transient posture interfaces. Users of these interfaces will seldom be technology-savvy and should, therefore, be presented the most simple and straightforward interfaces possible. These users are also accustomed to hardware controls. Unless an unprecedented ease of use can be achieved with a touch screen, dials and buttons (with appropriate audible feedback, and visual feedback via a view-only display or even hardware lamps) may be a better choice. Many appliance makers make the mistake of putting dozens of new — and unwanted — features into their new, digital models. Instead of making it easier, that "simple" LCD touchscreen becomes a confusing array of unworkable controls.

Another reason for a transient stance in appliance interfaces is that users of appliances are trying to get something very specific done. Like the users of transactional kiosks, they are not interested in exploring the interface or getting additional information; they simply want to put the washer on normal cycle or cook their frozen dinners.
One aspect of appliance design demands a different posture: Status information indicating what cycle the washer is on or what the VCR is set to record should be provided as a daemonic icon, providing minimal status quietly in a corner. If more than minimal status is required, an auxiliary posture for this information then becomes appropriate.

27.4 Flow and Transparency

When people are able to concentrate wholeheartedly on an activity, they lose awareness of peripheral problems and distractions. The state is called flow, a concept first identified by Mihaly Csikszentmihalyi, professor of psychology at the University of Chicago, and author of Flow: The Psychology of Optimal Experience (HarperCollins, 1991).

In People ware: Productive Projects and Teams (Dorset House, 1987), Tom DeMarco and Timothy Lister, describe flow as a “condition of deep, nearly meditative involvement.” Flow often induces a “gentle sense of euphoria” and can make you unaware of the passage of time. Most significantly, a person in a state of flow can be extremely productive, especially when engaged in process-oriented tasks such as "engineering, design, development, and writing." Today, these tasks are typically performed on computers while interacting with software. Therefore, it behooves us to create a software interaction that promotes and enhances flow, rather than one that includes potentially flow-breaking or flow-disturbing behavior. If the program consistently rattles the user out of flow, it becomes difficult for him to regain that productive state.

© Copyright Virtual University of Pakistan

If the user could achieve his goals magically, without your program, he would. By the same token, if the user needed the program but could achieve his goals without going through its user interface, he would. Interacting with software is not an aesthetic experience (except perhaps in games, entertainment, and exploration-oriented interactive systems). For the most part, it is a pragmatic exercise that is best kept to a minimum.

Directing your attention to the interaction itself puts the emphasis on the side effects of the tools rather than on the user's goals. A user interface is an artifact, not directly related to the goals of the user. Next time you find yourself crowing about what cool interaction you've designed, just remember that the ultimate user interface for most purposes is no interface at all.

To create flow, our interaction with software must become transparent. In other words, the interface must not call attention to itself as a visual artifact, but must instead, at every turn, be at the service of the user, providing what he needs at the right time and in the right place. There are several excellent ways to make our interfaces recede into invisibility. They are:

1. Follow mental models.
2. Direct, don't discuss.
3. Keep tools close at hand.
4. Provide modeless feedback.

We will now discuss each of these methods in detail.

Follow mental models

We introduced the concept of user mental models in previous lectures. Different users will have different mental models of a process, but they will rarely visualize them in terms of the detailed innards of the computer process. Each user naturally forms a mental image about how the software performs its task. The mind looks for some pattern of cause and effect to gain insight into the machine's behavior.

For example, in a hospital information system, the physicians and nurses have a mental model of patient information that derives from the patient records that they are used to manipulating in the real world. It therefore makes most sense to find patient information by using names of patients as an index. Each physician has certain patients, so it makes additional sense to filter the patients in the clinical interface so that each physician can choose from a list of her own patients, organized alphabetically by name. On the other hand, in the business office of the hospital, the clerks there are worried about overdue bills. They don't initially think about these bills in terms of who or what the bill is for, but rather in terms of how late the bill is (and perhaps how big the bill is). Thus, for the business office interface, it makes sense to sort first by time overdue and perhaps by amount due, with patient names as a secondary organizational principle.

Direct don't discuss

Many developers imagine the ideal interface to be a two-way conversation with the user. However, most users don't see it that way. Most users would rather interact with the software in the same way they interact with, say, their cars. They open the door and get in when they want to go somewhere. They step on the accelerator when they want the car to

© Copyright Virtual University of Pakistan

move forward and the brake when it is time to stop; they turn the wheel when they want the car to turn.

This ideal interaction is not a dialog — it's more like using a tool. When a carpenter hits nails, he doesn't discuss the nail with the hammer; he directs the hammer onto the nail. In a car, the driver — the user — gives the car direction when he wants to change the car's behavior. The driver expects direct feedback from the car and its environment in terms appropriate to the device: the view out the windshield, the readings on the various gauges on the dashboard, the sound of rushing air and tires on pavement, the feel of lateral g-forces and vibration from the road. The carpenter expects similar feedback: the feel of the nail sinking, the sound of the steel striking steel, and the heft of the hammer's weight.

The driver certainly doesn't expect the car to interrogate him with a dialog box. One of the reasons software often aggravates users is that it doesn't act like a car or a hammer. Instead, it has the temerity to try to engage us in a dialog — to inform us of our shortcomings and to demand answers from us. From the user's point of view, the roles are reversed: It should be the user doing the demanding and the software doing the answering.

With direct manipulation, we can point to what we want. If we want to move an object from A to B, we click on it and drag it there. As a general rule, the better, more flow-inducing interfaces are those with plentiful and sophisticated direct manipulation idioms.

Keep tools close at hand

Most programs are too complex for one mode of direct manipulation to cover all their features. Consequently, most programs offer a set of different tools to the user. These tools are really different modes of behavior that the program enters. Offering tools is a compromise with complexity, but we can still do a lot to make tool manipulation easy and to prevent it from disturbing flow. Mainly, we must ensure that tool information is plentiful and easy to see and attempt to make transitions between tools quick and simple. -Tools should be close at hand, preferably on palettes or toolbars. This way, the user can see them easily and can select them with a single click. If the user must divert his attention from the application to search out a tool, his concentration will be broken. It's as if he had to get up from his desk and wander down the hall to find a pencil. He should never have to put tools away manually.

Modeless feedback

As we manipulate tools, it's usually desirable for the program to report on their status, and on the status of the data we are manipulating with the tool. This information needs to be clearly posted and easy to see without obscuring or stopping the action.

When the program has information or feedback for the user, it has several ways to present it. The most common method is to pop up a dialog box on the screen. This technique is modal: It puts the program into a mode that must be dealt with before it can return to its normal state, and before the user can continue with her task. A better way to inform the user is with modeless feedback.

Feedback is modeless whenever information for the user is built into the main interface and doesn't stop the normal flow of system activities and interaction. In Word, you can

© Copyright Virtual University of Pakistan

see what page you are on, what section you are in, how many pages are in the current document, what position the cursor is in, and what time it is modelessly just by looking at the status bar at the bottom of the screen.

If you want to know how many words are in your document, however, you have to call up the Word Count dialog from the Tools menu. For people writing magazine articles, who need to be careful about word count, this information would be better delivered modeless. Jet fighters have a heads-up display, or HUD, that superimposes the readings of critical instrumentation onto the forward view of the cockpit's windscreen. The pilot doesn't even have to use peripheral vision, but can read vital gauges while keeping his eyes glued on the opposing fighter.

Our software should display information like a jet fighter's HUD. The program could use the edges of the display screen to show the user information about activity in the main work area of applications. Many drawing applications, such as Adobe Photoshop, already provide ruler guides, thumbnail maps, and other modeless feedback in the periphery of their windows.

27.5 Orchestration

When a novelist writes well, the craft of the writer becomes invisible, and the reader sees the story and characters with clarity undisturbed by the technique of the writer. Likewise, when a program interacts well with a user, the interaction mechanics precipitate out, leaving the user face-to-face with his objectives, unaware of the intervening software. The poor writer is a visible writer, and a poor interaction designer looms with a clumsily visible presence in his software.

To a novelist, there is no such thing as a "good" sentence. There are no rules for the way sentences should be constructed to be transparent. It all depends on what the protagonist is doing, or the effect the author wants to create. The writer knows not to insert an obscure word in a particularly quiet and sensitive passage, lest it sound like a sour note in a string quartet. The same goes for software. The interaction designer must train his ears to hear sour notes in the orchestration of software interaction. It is vital that all the elements in an interface work coherently together towards a single goal. When a program's communication with the user is well orchestrated, it becomes almost invisible.
Webster defines orchestration as "harmonious organization," a reasonable phrase for what we should expect from interacting with software. Harmonious organization doesn't yield to fixed rules. You can't create guidelines like, "Five buttons on a dialog box are good" and "Seven buttons on a dialog box are too many." Yet it is easy to see that a dialog box with 35 buttons is probably to be avoided. The major difficulty with such analysis is that it treats the problem in vitro. It doesn't take into account the problem being solved; it doesn't take into account what the user is doing at the time or what he is trying to accomplish.

Adding finesse: Less is more

For many things, more is better. In the world of interface design, the contrary is true, and we should constantly strive to reduce the number of elements in the interface without reducing the power of the system. In order to do this, we must do more with less; this is where careful orchestration becomes important. We must coordinate and control all the

© Copyright Virtual University of Pakistan

power of the product without letting the interface become a gaggle of windows and dialogs, covered with a scattering of unrelated and rarely used controls.

It's easy to create interfaces that are complex but not very powerful. They typically allow the user to perform a single task without providing access to related tasks. For example, most desktop software allows the user to name and save a data file, but they never let him delete, rename, or make a copy of that file at the same time. The dialog leaves that task to the operating system. It may not be trivial to add these functions, but isn't it better that the programmer perform the non-trivial activities than that the user to be forced to? Today, if the user wants to do something simple, like edit a copy of a file, he must go through a non-trivial sequence of actions: going to the desktop, selecting the file, requesting a copy from the menu, changing its name, and then opening the new file. Why not streamline this interaction?

It's not as difficult as it looks. Orchestration doesn't mean bulldozing your way through problems; it means finessing them wherever possible. Instead of adding the File Copy and Rename functions to the File Open dialog box of every application, why not just discard the File Open dialog box from every application and replace it with the shell program itself? When the user wants to open a file, the program calls the shell, which conveniently has all those collateral file manipulation functions built-in, and the user can double-click on the desired document. True, the application's File Open dialog does show the user a filtered view of files (usually limited to those formats recognized by the application), but why not add that functionality to the shell —filter by type in addition to sort by type?

Following this logic, we can also dispense with the Save As ... dialog, which is really the logical inverse of the File Open dialog. If every time we invoked the Save As ... function from our application, it wrote our file out to a temporary directory under some reasonable temporary name and then transferred control to the shell, we'd have all the shell tools at our disposal to move things around or rename them.

Yes, there would be a chunk of code that programmers would have to create to make it all seamless, but look at the upside. Countless dialog boxes could be completely discarded, and the user interfaces of thousands of programs would become more visually and functionally consistent, all with a single design stroke. That is finesse!

Distinguishing possibility from probability

There are many cases where interaction, usually in the form of a dialog box, slips into a user interface unnecessarily. A frequent source for such clinkers is when a program is faced with a choice. That's because programmers tend to resolve choices from the standpoint of logic, and it carries over to their software design. To a logician, if a proposition is true 999,999 times out of a million and false one time, the proposition is false — that's the way Boolean logic works. However, to the rest of us, the proposition is overwhelmingly true. The proposition has a possibility of being false, but the probability of it being false is minuscule to the point of irrelevancy. One of the most potent methods for better orchestrating your user interfaces is segregating the possible from the probable.
Programmers tend to view possibilities as being the same as probabilities. For example, a user has the choice of ending the program and saving his work, or ending the program and throwing away the document he has been working on for the last six hours.

© Copyright Virtual University of Pakistan

Mathematically, either of these choices is equally possible. Conversely, the probability of the user discarding his work is, at least, a thousand to one against; yet the typical program always includes a dialog box asking the user if he wants to save his changes.

The dialog box is inappropriate and unnecessary. How often do you choose to abandon changes you make to a document? This dialog is tantamount to your spouse telling you not to spill soup on your shirt every time you eat.

Providing comparisons

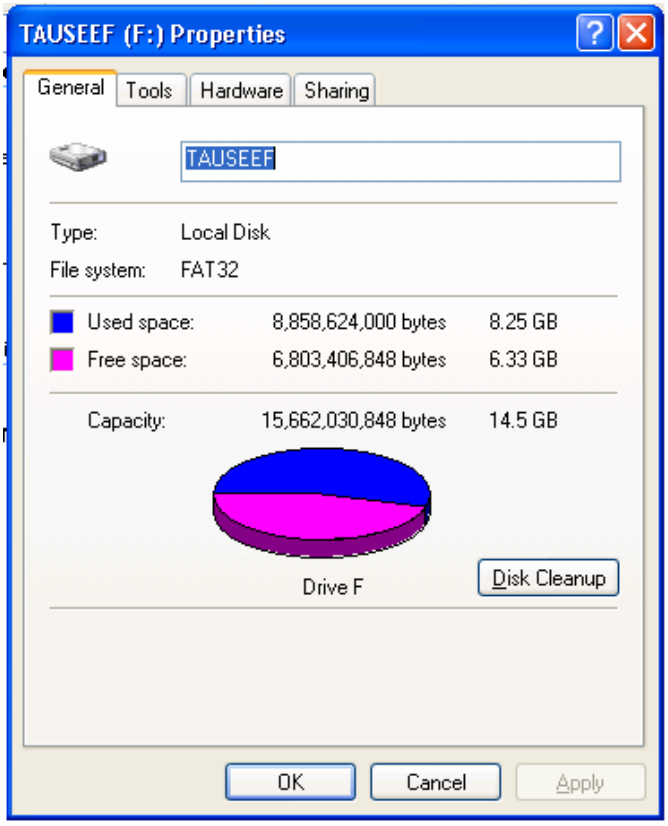
The way that a program represents information is another way that it can obtrude noisily into a user's consciousness. One area frequently abused is the representation of quantitative, or numeric, information. If an application needs to show the amount of free space on disk, it can do what the Microsoft Windows 3.x File Manager program did: give you the exact number of free bytes.

In the lower-left corner, the program tells us the number of free bytes and the total number of bytes on the disk. These numbers are hard to read and hard to interpret. With more than ten thousand million bytes of disk storage, it-ceases to be important to us just how many hundreds are left, yet the display rigorously shows us down to the kilobyte. But even while the program is telling us the state of our disk with precision, it is failing to communicate. What we really need to know is whether j or not the disk is getting full, or whether we can add a new 20 MB program and still have sufficient working room. These raw numbers, precise as they are, do little to help make sense of the facts.

Visual presentation expert Edward Tufte says that quantitative presentation should answer the question, "Compared to what?" Knowing that 231,728 KB are free on your hard disk is less useful than knowing that it is 22 percent of the disk's total capacity. Another Tufte dictum is, “Show the data,” rather than simply telling about it textually or numerically. A pie chart showing the used and unused portions in different colors would make it much easier to comprehend the scale and proportion of hard disk use. It would show us what 231,728 KB really means. The numbers shouldn't go away, but they should be relegated to the status of labels on the display and not be the display itself. They should also be shown with more reasonable and consistent precision. The meaning of the information could be shown visually, and the numbers would merely add support.

In Windows XP, Microsoft's right hand given whiles its left hand taken away. The Pile Manager is long dead, replaced by the Explorer dialog box shown in Figure.

© Copyright Virtual University of Pakistan



This replacement is the properties dialog associated with a hard disk. The Used Space is shown in blue and the Free Space is shown in magenta, making the pie chart an easy read. Now you can see at a glance the glad news that GranFromage is mostly empty.

Unfortunately, that pie chart isn't built into the Explorer's interface. Instead, you have to seek it out with a menu item. To see how full a disk is, you must bring up a modal dialog box that, although it gives you the information, takes you away from the place where you need to know it. The Explorer is where you can see, copy, move, and delete files; but it's not where you can easily see if things need to be deleted. That pie chart should have been built into the face of the Explorer. In Windows 2000, it is shown on the left-hand side when you select a disk in an Explorer window. In XP, however, Microsoft took a step backwards, and the graphic has once again been relegated to a dialog. It really should be visible at all times in the Explorer, along with the numerical data, unless the user chooses to hide it.

Using graphical input

Software frequently fails to present numerical information in a graphical way. Even rarer is the capability of software to enable graphical input. A lot of software lets users enter numbers; then, on command, it converts those numbers into a graph. Few products let the user enter a graph and, on command, convert that graph into a vector of numbers. By contrast, most modern word processors let you set tabs and indentations by dragging a marker on a ruler. The user can say, in effect, "Here is where I want the paragraph to start," and let the program calculate that it is precisely 1.347 inches in from the left margin instead of forcing the user to enter 1.347.

© Copyright Virtual University of Pakistan

"Intelligent" drawing programs like Microsoft Visio are getting better at this. Each polygon that the user manipulates on screen is represented behind the scenes by a small spreadsheet, with a row for each point and a column each for the X and Y coordinates. Dragging a polygon's vertex on screen causes the values in the corresponding point in the spreadsheet represented by the X and Y values, to change. The user can access the shape either graphically or through its spreadsheet representation.

This principle applies in a variety of situations. When items in a list need to be reordered, the user may want them ordered alphabetically, but he may also want them in order of personal preference; something no algorithm can offer. The user should be able to drag the items into the desired order directly, without an algorithm interfering with this fundamental operation.

Reflecting program status

When someone is asleep, he usually looks asleep. When someone is awake, he looks awake. When someone is busy, he looks busy: His eyes are focused on his work and his body language is closed and preoccupied. When someone is unoccupied, he looks unoccupied: His body is open and moving; his eyes are questing and willing to make contact. People not only expect this kind of subtle feedback from each other, they depend on it for maintaining social order.

Our programs should work the same way. When a program is asleep, it should look asleep. When a program is awake, it should look awake; and when it's busy, it should look busy. When the computer is engaged in some significant internal action like formatting a diskette, we should see some significant external action, such as the icon of the diskette slowly changing from grayed to active state. When the computer is sending a fax, we should see a small representation of the fax being scanned and sent (or at least a modeless progress bar). If the program is waiting for a response from a remote database, it should visually change to reflect its somnolent state. Program state is best communicated using forms of rich modeless feedback.

Avoiding unnecessary reporting

For programmers, it is important to know exactly what is happening process-wise in a program. This goes along with being able to control all the details of the process. For users, it is disconcerting to know all the details of what is happening. Non-technical people may be alarmed to hear that the database has been modified, for example. It is better for the program to just do what has to be done, issue reassuring clues when all is well, and not burden the user with the trivia of how it was accomplished.

Many programs are quick to keep users apprised of the details of their progress even though the user has no idea what to make of this information. Programs pop up dialog boxes telling us that connections have been made, that records have been posted, that users have logged on, that transactions were recorded, that data have been transferred, and other useless factoids. To software engineers, these messages are equivalent to the humming of the machinery, the babbling of the brook, the white noise of the waves crashing on the beach: They tell us that all is well. They were, in fact, probably used while debugging the software. To the user, however, these reports can be like eerie lights beyond the horizon, like screams in the night, like unattended objects flying about the room.

© Copyright Virtual University of Pakistan

As discussed before, the program should make clear that it is working hard, but the detailed feedback can be offered in a more subtle way. In particular, reporting information like this with a modal dialog box brings the interaction to a stop for no particular benefit.

It is important that we not stop the proceedings to report normalcy. When some event has transpired that was supposed to have transpired, never report this fact with a dialog box. Save dialogs for events that are outside of the normal course of events.

By the same token, don't stop the proceedings and bother the user with problems that are not serious. If the program is having trouble getting through a busy signal, don't put up a dialog box to report it. Instead, build a status indicator into the program so the problem is clear to the interested user but is not obtrusive to the user who is busy elsewhere.

The key to orchestrating the user interaction is to take a goal-directed approach. You must ask yourself whether a particular interaction moves the user rapidly and directly to his goal. Contemporary programs are often reluctant to take any forward motion without the user directing it in advance. But users would rather see the program take some "good enough" first step and then adjust it to what is desired. This way, the program has moved the user closer to his goal.

Avoiding blank slates

It's easy to assume nothing about what your users want and rather ask a bunch of questions of the user up front to help determine what they want. How many programs have you seen that start with a big dialog asking a bunch of questions? But users — not power users, but normal people — are very uncomfortable with explaining to a program what they want. They would much rather see what the program thinks is right and then manipulate that to make it exactly right. In most cases, your program can make a fairly correct assumption based on past experience. For example, when you create a new document in Microsoft Word, the program creates a blank document with preset margins and other attributes rather than opening a dialog that asks you to specify every detail. PowerPoint does a less adequate job, asking you to choose the base style for a new presentation each time you create one. Both programs could do better by remembering frequently and recently used styles or templates, and making those the defaults for new documents.

Just because we use the word think in conjunction with a program doesn't mean that the software needs to be intelligent (in the human sense) and try to determine the right thing to do by reasoning. Instead, it should simply do something that has a statistically good chance of being correct, then provide the user with powerful tools for shaping that first attempt, instead of merely giving the user a blank slate and challenging him
to have at it. This way the program isn't asking for permission to act, but rather asking for forgiveness after the fact.

For most people, a completely blank slate is a difficult starting point. It's so much easier to begin where someone has already left off. A user can easily fine-tune an approximation provided by the program into precisely what he desires with less risk of exposure and mental effort than he would have from drafting it from nothing.

© Copyright Virtual University of Pakistan

Command invocation versus configuration

Another problem crops up quite frequently, whenever functions with many parameters are invoked by users. The problem comes from the lack of differentiation between a function and the configuration of that function. If you ask a program to perform a function itself, the program should simply perform that function and not interrogate you about your precise configuration details. To express precise demands to the program, you would request the configuration dialog. For example, when you ask many programs to print a document, they respond by launching a complex dialog box demanding that you specify how many copies to print, what the paper orientation is, what paper feeder to use, what margins to set, whether the output should be in monochrome or color, what scale to print it at, whether to use Postscript fonts or native fonts, whether to print the current page, the current selection, or the entire document, and whether to print to a file and if so, how to name that file. All those options are useful, but all we wanted was to print the document, and that is all we thought we asked for.

A much more reasonable design would be to have a command to print and another command for print setup. The print command would not issue any dialog, but would just go ahead and print, either using previous settings or standard, vanilla settings. The print setup function would offer up all those choices about paper and copies and fonts. It would also be very reasonable to be able to go directly from the configure dialog to printing.

The print control on the Word toolbar offers immediate printing without a dialog box. This is perfect for many users, but for those with multiple printers or printers on a network, it may offer too little information. The user may want to see which printer is selected before he either clicks the control or summons the dialog to change it first. This is a good candidate for some simple modeless output placed on a toolbar or status bar (it is currently provided in the ToolTip for the control, which is good, but the feedback could be better still). Word's print setup dialog is called Print and is available from the File menu. Its name could be clearer, although the ellipsis does, according to GUI standards, give some inkling that it will launch a dialog.

There is a big difference between configuring and invoking a function. The former may include the latter, but the latter shouldn't include the former. In general, any user invokes a command ten times for every one time he configures it. It is better to make the user ask explicitly for configuration one time in ten than it is to make the user reject the configuration interface nine times in ten.

Microsoft's printing solution is a reasonable rule of thumb. Put immediate access to functions on buttons in the toolbar and put access to function-configuration dialog boxes on menu items. The configuration dialogs are better pedagogic tools, whereas the buttons provide immediate action.

Asking questions versus providing choices

Asking questions is quite different from providing choices. The difference between them is the same as that between browsing in a store and conducting a job interview. The individual asking the questions is understood to be in a position superior to the individual being asked. Those with authority ask questions; subordinates respond. Asking users questions make them feel inferior.

© Copyright Virtual University of Pakistan

Dialog boxes (confirmation dialogs in particular) ask questions. Toolbars offer choices. The confirmation dialog stops the proceedings, demands an answer, and it won't leave until it gets what it wants Toolbars, on the other hand, are always there, quietly and politely offering up their wares like a well-appointed store, offering you the luxury of selecting what you would like with just a flick of your finger.

Contrary to what many software developers think, questions and choices don't necessarily make the user feel empowered. More commonly, it makes the user feel badgered and harassed. Would you like soup or salad? Salad. Would you like cabbage or spinach? Spinach. Would you like French, Thousand Island, or Italian? French. Would you like local or regular? Stop! Just bring me the soup! Would you like chowder or chicken noodle? Users don't like to be asked questions. It cues the user that the program is:

- Ignorant
- Forgetful
- Weak
- Lacking initiative
- Unable to fend for itself
- Fretful
- Overly demanding

These are qualities that we typically dislike in people. Why should we desire them in software? The program is not asking us our opinion out of intellectual curiosity or desire to make conversation, the way a friend might over dinner. Rather, it is behaving ignorantly or presenting itself with false authority. The program isn't interested in our opinions; it requires information — often information it didn't really need to ask us in the first.

Worse than single questions are questions that is asked repeatedly and unnecessarily. Do you want to save that file? Do you want to save that file now? Do you really want to save that file? Software that asks fewer questions appears smarter to the user, and more polite, because if users fail to know the answer to a question, they then feel stupid.

In The Media Equation (Cambridge University Press, 1996), Stanford sociologists Clifford Nass and Byron Reeves make a compelling case that humans treat and respond to computers and other interactive products as if they were people. We should thus pay real attention to the "personality" projected by our software. Is it quietly competent and helpful, or does it whine, nag, badger, and make excuses?

Choices are important, but there is a difference between being free to make choices based on presented information and being interrogated by the program in modal fashion. Users would much rather direct their software the way they direct their automobiles down the street. An automobile offers the user sophisticated choices without once issuing a dialog box.

Hiding ejector seat levers

In the cockpit of every jet fighter is a brightly painted lever that, when pulled, fires a small rocket engine underneath the pilot's seat, blowing the pilot, still in his seat, out of

© Copyright Virtual University of Pakistan

the aircraft to parachute safely to earth. Ejector seat levers can only be used once, and their consequences are significant and irreversible.

Just like a jet fighter needs an ejector seat lever, complex desktop applications need configuration facilities. The vagaries of business and the demands placed on the software force it to adapt to specific situations, and it had better be able to do so. Companies that pay millions of dollars for custom software or site licenses for thousands of copies of shrink-wrapped products will not take kindly to a program's inability to adapt to the way things are done in that particular company. The program must adapt, but such adaptation can be considered a one-time procedure, or something done only by the corporate IT staff on rare occasion. In other words, ejector seat levers may need to be used, but they won't be used very often.

Programs must have ejector seat levers so that users can — occasionally — move persistent objects in the interface, or dramatically (sometimes irreversibly) alter the function or behavior of the application. The one thing that must never happen is accidental deployment of the ejector seat. The interface design must assure that the user can never inadvertently fire the ejector seat when all he wants to do is make some minor adjustment to the program.

Ejector seat levers come in two basic varieties: those that cause a significant visual dislocation (large changes in the layout of tools and work areas) in the program, and those that perform some irreversible action. Both of these functions should be hidden from inexperienced users. Of the two, the latter variety is by far the more dangerous. In the former, the user may be surprised and dismayed at what happens next, but he can at least back out of it with some work. In the latter case, he and his colleagues are likely to be stuck with the consequences.

© Copyright Virtual University of Pakistan

Lecture 28.

Behavior & Form Part III

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the narratives and scenarios
- Define requirements using persona-based design

28.1 Eliminating Excise

Software too often contains interactions that are top-heavy with extra work for the user Programmers typically focus so intently on the enabling technology that they don't carefully consider the human actions required to operate the technology from a goal-directed point-of-view. The result is software that charges its users a tax, or excise, of cognitive and sometimes even physical effort every time it is used. This lecture focuses on the nature of this excise, and discusses the means by which it can be reduced and even eliminated altogether.

What Is Excise?

When we decide to drive to the office, we must open the garage door, get in, start the motor, back out, and close the garage door before we even begin the forward motion that will take us to our destination. All these actions are in support of the automobile rather than in support of getting to the destination. If we had Star Trek transporters instead, we'd dial up our destination coordinates and appear there instantaneously — no garages, no motors, no traffic lights. Our point is not to complain about the intricacies of driving, but rather to distinguish between two types of actions we take to accomplish our daily tasks.

Any large task, such as driving to the office, involves many smaller tasks. Some of these tasks work directly toward achieving the goal; these are tasks like steering down the road toward your office. Excise tasks, on the other hand, don't contribute directly to reaching the goal, but are necessary to accomplishing it just the same. Such tasks include opening and closing the garage door, starting the engine, and stopping at traffic lights, in addition to putting oil and gas in the car and performing periodic maintenance.

Excise is the extra work that satisfies either the needs of our tools or those of outside agents as we try to achieve our objectives. The distinction is sometimes hard to see because we get so used to the excise being part of our tasks. Most of us drive so frequently that differentiating the act of opening the garage door from the act of driving towards the destination is difficult. Manipulating the garage door is something we do for the car, not for us, and it doesn't move us towards our destination the way the accelerator pedal and steering wheel do. Stopping at red lights is something imposed on us by our society that, again, doesn't help us achieve our true goal. (In this case, it does help us achieve a related goal of arriving safely at our office.)

Software, too, has a pretty clear dividing line between goal-directed tasks and excise tasks. Like automobiles, some software excise tasks are trivial, and performing them is no great hardship. On the other hand, some software excise tasks are as obnoxious as fixing a

© Copyright Virtual University of Pakistan

flat tire. Installation leaps to mind here, as do such excise tasks as configuring networks, making backups, and connecting to online services.

The problem with excise tasks is that the effort we expend in doing them doesn't go directly towards accomplishing our goals. Where we can eliminate the need for excise tasks, we make the user more effective and productive and improve the usability of the software. As a software designer, you should become sensitive to the presence of excise and take steps to eradicate it with the same enthusiasm a doctor would apply to curing an infection.

There are many such instances of petty excise, particularly in GUIs. Virtually all window management falls into this category. Dragging, reshaping, resizing, reordering, tiling and cascading windows qualify as excise actions.

GUI Excise

One of the main criticisms leveled at graphical user interfaces by experienced computer users — notably those trained on command-line systems — is that getting to where you want to go is made slower and more difficult by the extra effort that goes into manipulating windows and icons. Users complain that, with a command line, they can just type in the desired command and the computer executes it immediately. With windowing systems, they must open various folders looking for the desired file or program before they can launch it. Then, after it appears on the screen, they must stretch and drag the window until it is in the desired location and configuration.

These complaints are well founded. Extra window manipulation tasks like these are, indeed, excise. They don't move the user towards his goal; they are overhead that the programs demand before they deign to assist the user. But everybody knows that GUIs are easier to use than command-line systems. Who is right?

The confusion arises because the real issues are hidden. The command-line interface forces an even more expensive excise budget on the user: He must first memorize the commands. Also, he cannot easily configure his screen to his own personal requirements. The excise of the command-line interface becomes smaller only after the user has invested significant time and effort in learning it.

On the other hand, for the casual or first-time user, the visual explicitness of the GUI helps him navigate and learn what tasks are appropriate and when. The step-by-step nature of the GUI is a great help to users who aren't yet familiar with the task or the system. It also benefits those users who have more than one task to perform and who must use more than one program at a time.

Excise and expert users

Any user willing to learn a command-line interface automatically qualifies as a power user. And any power user of a command-line interface will quickly become a power user of any other type of interface, GUI included. These users will easily learn each nuance of the programs they use. They will start up each program with a clear idea of exactly what it is they want to do and how they want to do it. To this user, the assistance offered to the casual or first-time user is just in the way.

We must be careful when we eliminate excise. We must not remove it just to suit power users. Similarly, however, we must not force power users to pay the full price of our providing help to new or infrequent users.

© Copyright Virtual University of Pakistan

Training wheels

One of the areas where software designers can inadvertently introduce significant amounts of excise is in support for first-time or casual users. It is easy to justify adding facilities to a program that will make it easy for newer users to learn how to use the program. Unfortunately, these facilities quickly become excise as the users become familiar with the program — perpetual intermediates. Facilities added to software for the purpose of training beginners must be easily turned off. Training wheels are rarely needed for extended periods of time, and training wheels, although they are a boon to beginners, are a hindrance to advanced learning and use when they are left on permanently.

"Pure" excise

There are a number of actions that are excise of such purity that nobody needs them, from power users to first-timers. These include most hardware-management tasks that the computer could handle itself, like telling a program which COM port to use. Any demands for such information should be struck from user interfaces and replaced with more intelligent program behavior behind the scenes.

Visual excise

Designers sometimes paint themselves into excise corners by relying too heavily on visual metaphors. Visual metaphors like desktops with telephones, copy machines, staplers, and fax machines — or file cabinets with folders in drawers — are cases in point. These visual metaphors may make it easy to understand the relationships between program elements and behaviors; but after these fundamentals are learned, the management of the metaphor becomes pure excise. In addition, the screen space consumed by the images becomes increasingly egregious, particularly in sovereign posture applications. The more we stare at the program from day to day, the more we resent the number of pixels it takes to tell us what we already know. The little telephone that so charmingly told us how to dial on that first day long ago is now a barrier to quick communications.

Transient posture applications can tolerate more training and explanation excise than sovereign applications. Transient posture programs aren't used frequently, so their users need more assistance in understanding what the program does and remembering how to control it. For sovereign posture applications, however, the slightest excise becomes agonizing over time.

The second type of visual excise was not a significant issue before the advent of the Web: overemphasis of visual design elements to the extent that they interfere with user goals and comprehension. The late 90s attracted a large number of graphic and new media designers to the Web, people who viewed this medium as a predominantly visual one, the experience of which was defined by rich, often animated, visuals. Although this might have been (and perhaps still is) appropriate for brochure-ware Web sites that serve primarily as marketing collateral, it is highly inappropriate for transactional Web sites and Web applications. As we will discuss more in coming lectures, these latter types of sites, into which the majority of e-commerce falls, have far more in common, from a behavioral standpoint, with sovereign desktop applications than with multimedia kiosk-ware or brochure-ware.

The result was that many visually arresting, visually innovative sites were spawned that ignored the two most critical elements: an understanding of user goals and a streamlined

© Copyright Virtual University of Pakistan

behavior that helped users achieve them. A pre-eminent example of this was Boo.com, one of the first major implosions of the dot.com bust. This fashion e-tailor made use of hip visuals and flash-based interactive agents, but didn't seem to spend much effort addressing user goals. The site was sluggish due to flash, visually distracting, confusingly laid out, and difficult to navigate due to multiple windows and confusing links. Boo attempted to be high-concept, but its users' goals were simply to buy products more quickly, cheaply, and easily on-line than they could elsewhere. By the time some of these problems were remedied, Boo's customers had abandoned them. One can only wonder how much difference a goal-directed design might have made to Boo and many other e-commerce failures of that time.

Unfortunately, some of these visual excesses are slowly creeping into desktop applications, as programmers and designers borrow flashy but inappropriate idioms from the Web.

Determining what is excise

Sometimes we find certain tasks like window management, which, although they are mainly for the program, are useful for occasional users or users with special preferences. In this case, the function itself can only be considered excise if it is forced on the user rather than made available at his discretion.

The only way to determine whether a function or behavior is excise is by comparing it to the user's goals. If the user needs to see two programs at a time on the screen in order to compare or transfer information, the ability to configure the main windows of the programs so that they share the screen space is not excise. If the user doesn't have this specific goal, a requirement that the user must configure the main window of either program is excise.

28.2 Navigation and Inflection

Desktop applications, Web sites, and devices all have one particular attribute in common that, if improperly designed, becomes a critical obstacle to usability: navigation. The user must be able to navigate efficiently through the features and facilities of a program, Web site, or device. He must also be able to stay oriented in the program as he moves from screen to screen.

A user can navigate if he always understands what he has to do next, knows what state the program, site, or device is in, and knows how to find the tools he needs. This chapter discusses the issues surrounding navigation, and how to better help users navigate through interactive products.

Navigation Is Excise

As discussed earlier, the most important thing to realize about navigation is that, in almost all cases, it represents pure excise, or something close to it. Except in games where the goal is to navigate successfully through a maze of obstacles, navigating through software does not meet user goals, needs, or desires. Unnecessary or difficult

navigation thus becomes a major frustration to users. In fact, it is the authors' opinion that poorly designed navigation presents the number-one problem in the design of any software application or system — desktop, Web-based, or otherwise. It is also the place where the programmer's implementation model is made most apparent to the user.

© Copyright Virtual University of Pakistan

Types of Navigation

Navigation through software occurs at multiple levels. The following list enumerates the most common types of navigation:

- Navigation between multiple windows or screens
- Navigation between panes within a window (or frames in a page)
- Navigation between tools or menus in a pane
- Navigation within information displayed in a pane or frame (for example: scrolling, panning, zooming, following links)

Some students may question the inclusion of some of the bullets above as types of navigation. Broad definition of navigation are purposely being used: any action that takes the user to a new part of the interface or which requires him to otherwise locate objects, tools, or data. The reason for this is simple: When we start thinking about these actions as navigation, it becomes clear that they are excise and should, therefore, be minimized or eliminated. We'll now discuss each of these types of navigation in more detail.

Navigation between multiple windows or pages

Navigation between multiple windows is perhaps the most disorienting kind of navigation for users. Navigating between windows involves a gross shifting of attention that disrupts the users flow and forces him into a new context. The act of navigating to another window also often means that the contents of the original window are partly or completely obscured. At the very least, it means that the user needs to worry about window management, an excise task that further disrupts his flow. If users must constantly shuttle back and forth between windows to achieve their goals, their productivity will drop, and their disorientation and frustration levels will rise. If the number of windows is large enough, the user will become sufficiently disoriented that he may experience navigational trauma: He gets lost in the interface. Sovereign posture applications avoid this problem by placing all main interactions in a single primary window, which may contain multiple independent panes.

Navigation between panes

Windows can contain multiple panes, either adjacent to each other and separated by splitters or stacked on top of each other and denoted by tabs. Adjacent panes can solve many navigation problems by placing useful supporting functions, links, or data directly adjacent to the primary work or display area, thus reducing navigation to almost nil. If objects can be dragged between panes, those panes should be adjacent to each other.

Problems arise when adjacent supporting panes become too numerous, or when they are not placed on the screen in a way that matches the user's workflow. Too many adjacent panes result in visual clutter and confusion: The user does not know where to go to find what he needs. Also, crowding forces the introduction of scrolling, which is another navigational hit. Navigation within the single screen thus becomes a problem. Some Web portals, trying to be everything to everyone, have such navigational problems.

In some cases, depending on user workflows, tabbed panes can be appropriate. Tabbed panes bring with them a level of navigational excise and potential for disorientation because they obscure what was on the screen before the user navigated to them. However, this idiom is appropriate for the main work area when multiple documents or independent views of a document are required (such as in Microsoft Excel).

© Copyright Virtual University of Pakistan

Some programmers interpret tabs as permission to break complex facilities into smaller chunks and place one per pane. They reason that using these facilities will somehow become easier if the functionality is simply cut into pieces. Actually, by putting parts of a single facility onto separate panes, the excise increases, whereas the user's understanding and orientation decrease. What's more, doing this violates the axiom: A dialog box (or pop-up window) is another room: have a good reason to go there. Most users of most programs simply don't require that their software tools have dozens of controls for normal use.

Tabbed panes can be appropriate when there are multiple supporting panes for a work area that are not used at the same time. The support panes can then be stacked, and the user can choose the pane suitable for his current tasks, which is only a single click away. Microsoft Internet Explorer for the Macintosh uses a variant of these stacked panes. If no pane is selected (users can deselect by clicking on the active tab), the program shuts the adjacent pane like a drawer, leaving only the tabs visible. This variant is useful if space is at a premium.

Navigation between tools and menus

Another important and overlooked form of navigation results from the user's need to make use of different tools, palettes, and functions. Spatial organization of these within a pane or window is critical to minimizing extraneous mouse movements that, at best, could result in user annoyance and fatigue, and at worst, result in repetitive stress injury. Tools that are used frequently and in conjunction with each other should be grouped together spatially and also be immediately available. Menus require more navigational effort on the part of the user because their contents are not visible prior to clicking. Frequently used functions should be provided in toolbars, palettes, or the equivalent Menu use should be reserved only for infrequently accessed commands.

Adobe Photoshop 6.0 exhibits some annoying behaviors in the way it forces users to navigate between palette controls. For example, the Paint Bucket tool and the Gradient tool each occupy the same location on the tool palette; you must select between them by clicking and holding on the visible control, which opens a menu that lets you select between them. However, both are fill tools, and both are frequently used. It would have been better to place each of them on the palette next to each other to avoid that frequent, flow-disrupting tool navigation.

Navigation of information

Navigation of information, or of the content of panes or windows, can be accomplished by several methods: scrolling (panning), linking (jumping), and zooming. The first two methods are common: scrolling is ubiquitous in most software and linking is ubiquitous on the Web (though increasingly, linking idioms are being adopted in non-Web applications). Zooming is primarily used for visualization of 3D and detailed 2D data.

Scrolling is often a necessity, but the need for it should be minimized when possible. Often there is a tradeoff between paging and scrolling information: You should understand your users' mental models and workflows to determine what is best for them.

In 2D visualization and drawing applications, vertical and horizontal scrolling is common. These kinds of interfaces benefit from a thumbnail map to ease navigation.

Linking is the critical navigational paradigm of the Web. Because it is a visually dislocating activity, extra care must be taken to provide visual and textual cues that help orient users.

© Copyright Virtual University of Pakistan

Zooming and panning are navigational tools for exploring 2D and 3D information. These methods are appropriate when creating 2D or 3D drawings and models or for exploring representations of real-world 3D environments (architectural walkthroughs, for example). They typically fall short when used to examine arbitrary or abstract data presented in more than two dimensions. Some information visualization tools use zoom to mean, "display more attribute details about objects," a logical rather than spatial zoom. As the view of the object enlarges, attributes (often textual) appear superimposed over its graphical representation. This kind of interaction is almost always better served through an adjacent supporting pane that displays the properties of selected objects in a more standard, readable form. Users find spatial zoom difficult enough to understand; logical zoom is arcane to all but visualization researchers and the occasional programmer.
Panning and zooming, especially when paired together, create enormous navigation difficulties for users. Humans are not used to moving in unconstrained 3D space, and they have difficulty perceiving 3D properly when it is projected on a 2D screen (see Chapter 24 for more discussion of 3D manipulation).

Improving Navigation

There are many ways to begin improving (eliminating, reducing, or speeding) navigation in your applications, Web sites, and devices. Here are the most effective:

- Reduce the number of places to go
- Provide signposts
- Provide overviews
- Provide appropriate mapping of controls to functions
- Inflect your interface to match user needs
- Avoid hierarchies

We'll discuss these in detail:

Reduce the number of places to go

The most effective method of improving navigation sounds quite obvious: Reduce the number of places to which one must navigate. These "places" include modes, forms, dialogs, pages, windows, and screens. If the number of modes, pages, or screens is kept to a minimum, the user's ability to stay oriented increases dramatically. In terms of the four types of navigation presented earlier, this directive means:

- Keep the number of pages and windows to a minimum: One full-screen window with two or three views (maximum) is best. Keep dialogs, especially modeless dialogs, to a minimum. Programs or Web sites with dozens of distinct types of pages, screens, or forms are not navigable under any circumstances.
- Keep the number of adjacent panes in your window or Web page limited to the minimum number needed for users to achieve their goals. In sovereign applications, three panes is a good maximum. On Web pages, anything more than two navigation areas and one content area begins to get busy.
- Keep the number of controls limited to as few as your users really need to meet their goals. Having a good grasp of your users via personas will enable you to avoid functions and controls that your users don't really want or need and that, therefore, only get in their way.
- Scrolling should be minimized when possible. This means giving supporting panes enough room to display information so that they don't require constant

© Copyright Virtual University of Pakistan

scrolling. Default views of 2D and 3D diagrams and scenes should be such that the user can orient himself without too much panning around. Zooming, particularly continuous zooming, is the most difficult type of navigation for most users so its use should be discretionary, not a requirement.

Many e-commerce sites present confusing navigation because the designers are trying to serve everyone with one generic site. If a user buys books but never CDs from a site, access to the CD portion of the site could be de-emphasized in the main screen for that user. This makes more room for that user to buy books, and the navigation becomes simpler. Conversely, if he visits his account page frequently, his version of the site should have his account button (or tab) presented prominently.

Provide signposts

In addition to reducing the number of navigable places, another way to enhance the user's ability to find his way around is by providing better points of reference — signposts. In the same way that sailors navigate by reference to shorelines or stars, users navigate by reference to persistent objects placed in the user interface.

Persistent objects, in a desktop world, always include the program's windows. Each program most likely has a main, top-level window. The salient features of that window are also considered persistent objects: menu bars, toolbars, and other palettes
or visual features like status bars and rulers. Generally, each window of the program has a distinctive look that will soon become instantly recognizable.

On the Web, similar rules apply. The best Web applications, such as Amazon.com, make careful use of persistent objects that remain constant throughout the shopping experience, especially the tab bar along the top of the page and the Search and Browse areas on the left of the page. Not only do these areas provide clear navigational options, but their consistent presence and layout also help orient customers.

In devices, similar rules apply to screens, but hardware controls themselves can take on the role of signposts — even more so when they are able to offer visual or tactile feedback about their state. Radio buttons that, for example, light when selected, even a needle's position on a dial, can provide navigational information if integrated appropriately with the software.

Depending on the application, the contents of the program's main window may also be easily recognizable (especially true in kiosks and small-screen devices). Some programs may offer a few different views of their data, so the overall aspect of their screens will change depending on the view chosen. A desktop application's distinctive look, however, will usually come from its unique combination of menus, palettes, and toolbars. This means that menus and toolbars must be considered aids to navigation. You don't need a lot of signposts to navigate successfully. They just need to be visible. Needless to say, signposts can't aid navigation if they are removed, so it is best if they are permanent fixtures of the interface.

Making each page on a Web site look just like every other one may appeal to marketing, but it can, if carried too far, be disorienting. Certainly, you should use common elements consistently on each page, but by making different rooms look visually distinct, — that is making the purchase page look very different from the new account page — you will help to orient your users better.

© Copyright Virtual University of Pakistan

MENUS

The most prominent permanent object in a program is the main window and its title and menu bars. Part of the benefit of the menu comes from its reliability and consistency. Unexpected changes to a program's menus can deeply reduce the user's trust in them. This is true for menu items as well as for individual menus. It is okay to add items to the bottom of a menu, but the standard suite of items in the main part of it should change only for a clearly demonstrable need.

TOOLBARS

If the program has a toolbar, it should also be considered a recognizable signpost. Because toolbars are idioms for perpetual intermediates rather than for beginners, the strictures against changing menu items don't apply quite as strongly to individual toolbar controls. Removing the toolbar itself is certainly a dislocating change to a persistent object. Although the ability to do so should he there, it shouldn't be offered casually, and the user should be protected against accidentally triggering it. Some programs put controls on the toolbar that made the toolbar disappear! This is a completely inappropriate ejector seat lever.

OTHER INTERFACE SIGNPOSTS;

Tool palettes and fixed areas of the screen where data is displayed or edited should also be considered persistent objects that add to the navigational ease of the interface. Judicious use of white space and legible fonts is important so that these signposts remain clearly evident and distinct.

Provide overviews

Overviews serve a similar purpose to signposts in an interface: They help to orient the user. The difference is that overviews help orient users within the content rather than within the application as a whole. Because of this, the overview area should itself be persistent; its content is dependent on the data being navigated.

Overviews can be graphical or textual, depending on the nature of the content. An excellent example of a graphical overview is the aptly named Navigator palette in Adobe Photoshop.

In the Web world, the most common form of overview area is textual: the ubiquitous breadcrumb display. Again, most breadcrumbs provide not only a navigational aid, but a navigational control as well: They not only show where in the data structure the user is, but they give him tools to move to different nodes in the structure in the form of links.

A final interesting example of an overview tool is the annotated scrollbar. Annotated scrollbars are most useful for scrolling through text. They make clever use of the linear nature of both scrollbars and textual information to provide location information about the locations of selections, highlights, and potentially many other attributes of formatted or unformatted text. Hints about the locations of these items appear in the "track" that the thumb of the scrollbar moves in, at the appropriate location. When the thumb is over the annotation, the annotated feature of the text is visible in the display. Microsoft Word uses a variant of the annotated scrollbar; it shows the page number and nearest header in a ToolTip that remains active during the scroll.

© Copyright Virtual University of Pakistan

Provide appropriate mapping of controls to functions

Mapping describes the relationship between a control, the thing it affects, and the intended result. Poor mapping is evident when a control does not relate visually or symbolically with the object it affects. Poor mapping requires the user to stop and think about the relationship, breaking flow Poor mapping of controls to functions increases the cognitive load for users and can result in potentially serious user errors.

Inflect your interface to match user needs

Inflecting an interface means organizing it to minimize typical navigation. In practice, this means placing the most frequently desired functions and controls in the most immediate and convenient locations for the user to access them, while pushing the less frequently used functions deeper into the interface, where the user won't stumble over them. Rarely used facilities shouldn't be removed from the program, but they should be removed from the user's everyday workspace.

The most important principle in the proper inflection of interfaces is commensurate efforts. Although it applies to all users, it is particularly pertinent to perpetual intermediates. This principle merely states that people will willingly work harder for something that is more valuable to get. The catch, of course, is that value is in the eye of the beholder. It has nothing to do with how technically difficult a feature is to implement, but rather has entirely to do with the user's goals.

If the user really wants something, he will work harder to get it. If a person wants to become a good tennis player, for example, he will get out on the court and play very hard. To someone who doesn't like tennis, any amount of the sport is tedious effort. If a user needs to format beautiful documents with multiple columns, several fonts, and fancy headings to impress his boss, he will be highly motivated to explore the recesses of the program to learn how. He will be putting commensurate effort into the project. If some other user just wants to print plain old documents in one column and one font, no amount of inducement will get him to learn those more-advanced formatting features.

This means that if you add features to your program that are necessarily complex to manage, users will be willing to tolerate that complexity only if the rewards are worth it. This is why a program s user interface can't be complex to achieve simple results, but it can be complex to achieve complex results (as long as such results aren't needed very often).

It is acceptable from an interface perspective to make advanced features something that the user must expend a little extra effort to activate, whether that means searching in a menu, opening a dialog, or opening a drawer. The principle of commensurate effort allows us to inflect interfaces so that simple, commonly used functions are immediately at hand at all times. Advanced features, which are less frequently used but have a big payoff for the user, can be safely tucked away where they can be brought up only when needed. In general, controls and displays should be organized in an interface according to three attributes: frequency of use, degree of dislocation, and degree of exposure.

- Frequency of use means how often the controls, functions, objects, or displays are used in typical day-to-day patterns of use. Items and tools that are most frequently used (many times a day) should be immediately in reach. Less frequently used items, used perhaps once or twice a day, should be no more than a click or two away. Other items can be two or three clicks away.

© Copyright Virtual University of Pakistan

- Degree of dislocation refers to the amount of sudden change in an interface or in the document/information being processed by the application caused by the invocation of a specific function or command. Generally speaking, it's a good idea to put these types of functions deeper into the interface.
- Degree of exposure deals with functions that are irreversible or which may have other dangerous ramifications. ICBMs require two humans turning keys simultaneously on opposite sides of the room to arm them. As with dislocating functions, you want to make these types of functions more difficult for your users to stumble across.

Of course, as users get more experienced with these features, they will search for shortcuts, and you must provide them. When software follows commensurate effort, the learning curve doesn't go away, but it disappears from the user's mind — which is just as good.

© Copyright Virtual University of Pakistan

Lecture 29.

Evaluation – Part I

Learning Goals

The aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand what evaluation is in the development process
- Understand different evaluation paradigms and techniques

What to evaluate?

There is a huge variety of interactive products with a vast array of features that need to be evaluated. Some features, such as the sequence of links to be followed to find an item on a website, are often best evaluated in a laboratory, since such a setting allows the evaluators to control what they want to investigate. Other aspects, such as whether a collaborative toy is robust and whether children enjoy interacting with it, are better evaluated in natural settings, so that evaluators can see what children do when left to their own devices.

John Gould and his colleagues (Gould et aL 1990; Gould and Lewis, 1985) recommended three principles for developing the 1984 Olympic Message System:

- Focus on users and their tasks
- Observe, measure, and analyze their performance with the system
- Design lucratively

Since the OMS study, a number of new evaluation techniques have been developed. There has also been a growing trend towards observing how people interact with the system in their work, home, and other settings, the goal being to obtain a better understanding of how the product is (or will be) used in its intended setting. For example, at work people are frequently being interrupted by phone calls, others knocking at their door, email arriving, and so on—to the extent that many tasks are interrupt-driven. Only rarely does someone carry a task out from beginning to end without stopping to do something else. Hence the way people carry out an activity (e.g., preparing a report) in the real world is very different from how it may be observed in a laboratory. Furthermore, this observation has implications for the way products should be designed.

Why you need to evaluate?

Just as designers shouldn't assume that everyone is like them, they also shouldn't presume that following design guidelines guarantees good usability. Evaluation is needed to check that users can use the product and like it. Furthermore, nowadays users look for much more than just a usable system, as the Nielsen Norman Group, a usability consultancy company, point out (www.nngroup.com):

> "User experience" encompasses all aspects of the end-user's interaction ... the first requirement for an exemplary user experience is to meet the exact needs

© Copyright Virtual University of Pakistan

> of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use. "

Bruce Tognazzini another successful usability consultant, comments (www.asktog.com) that:

> "Iterative design, with its repeating cycle of design and testing, is the only validated methodology in existence that will consistently produce successful results. If you don't have user-testing as an integral part of your design process you are going to throw buckets of money down the drain."

Tognazzini points out that there are five good reasons for investing in user testing:
1. Problems are fixed before the product is shipped, not after.
2. The team can concentrate on real problems, not imaginary ones.
3. Engineers code instead of debating.
4. Time to market is sharply reduced.
5. Finally, upon first release, your sales department has a rock-solid design it can sell without having to pepper their pitches with how it will all actually work in release 1.1 or 2.0.

Now that there is a diversity of interactive products, it is not surprising that the range of features to be evaluated is very broad. For example, developers of a new web browser may want to know if users find items faster with their product. Government authorities may ask if a computerized system for controlling traffic lights results in fewer accidents. Makers of a toy may ask if six-year-olds can manipulate the controls and whether they are engaged by its furry case and pixie face. A company that develops the casing for cell phones may ask if the shape, size, and color of the case is appealing to teenagers. A new dotcom company may want to assess market reaction to its new home page design.

This diversity of interactive products, coupled with new user expectations, poses interesting challenges for evaluators, who, armed with many well tried and tested techniques, must now adapt them and develop new ones. As well as usability, user experience goals can be extremely important for a product's success.

When to evaluate?

The product being developed may be a brand-new product or an upgrade of an existing product. If the product is new, then considerable time is usually invested in market research. Designers often support this process by developing mockups of the potential product that are used to elicit reactions from potential users. As well as helping to assess market need, this activity contributes to understanding users' needs and early requirements. As we said in earlier lecture, sketches, screen mockups, and other low-fidelity prototyping techniques are used to represent design ideas. Many of these same techniques are used to elicit users" opinions in evaluation (e.g., questionnaires and interviews), but the purpose and focus of evaluation are different. The goal of evaluation is to assess how well a design fulfills users' needs and whether users like it.

In the case of an upgrade, there is limited scope for change and attention is focused on improving the overall product. This type of design is well suited to usability engineering in which evaluations compare user performance and attitudes with those for previous versions. Some products, such as office systems, go through many
versions, and successful products may reach double-digit version numbers. In contrast, new products do not have previous versions and there may be nothing comparable on the market, so more radical changes are possible if evaluation results indicate a problem.

© Copyright Virtual University of Pakistan

Evaluations done during design to check that the product continues to meet users' needs are known *as formative evaluations.* Evaluations that are done to assess the success of a finished product, such as those to satisfy a sponsoring agency or to check that a standard is being upheld, are known as *summative evaluation.* Agencies such as National Institute of Standards and Technology (NIST) in the USA, the International Standards Organization (ISO) and the British Standards Institute (BSI) set standards by which products produced by others are evaluated.

29.1 Evaluation paradigms and techniques

Before we describe the techniques used in evaluation studies, we shall start by proposing some key terms. Terminology in this field tends to be loose and often confusing so it is a good idea to be clear from the start what you mean. We start with the much-used term *user studies,* defined by Abigail Sellen in her interview as follows: "user studies essentially involve looking at how people behave either in their natural [environments], or in the laboratory, both with old technologies and with new ones." Any kind of evaluation, whether it is a user study or not, is guided either explicitly or implicitly by a set of beliefs that may also he underpinned by theory. These beliefs and the practices (i.e., the methods or techniques) associated with them are known as an evaluation paradigm, which you should not confuse with the "interaction paradigms. Often evaluation paradigms are related to a particular discipline in that they strongly influence how people from the discipline think about evaluation. Each paradigm has particular methods and techniques associated with it. So that you are not confused, we want to state explicitly that we will not be distinguishing between methods and techniques. We tend to talk about techniques, but you may find that other some call them methods. An example of the relationship between a paradigm and the techniques used by evaluators following that paradigm can be seen for usability testing, which is an applied science and engineering paradigm. The techniques associated wild usability testing are: user testing in a controlled environment; observation of user activity in the controlled environment and the field; and questionnaires and interviews.

Evaluation paradigms

In this lecture we identify four core evaluation paradigms: (1) “quick and dirty” evaluations; (2) usability testing; (3) field studies; and (4) predictive evaluation. Other people may use slightly different terms to refer to similar paradigms.

"Quick and dirty" evaluation

A "quick and dirty" evaluation is a common practice in which designers informally get feedback from users or consultants to confirm that their ideas are in line with users" needs and are liked. "Quick and dirty" evaluations can be done at any stage and the emphasis is on fast input rather than carefully documented findings. For example, early in design developers may meet informally with users to get feedback on ideas
for a new product (Hughes el al., 1994). At later stages similar meetings may occur to try out an idea for an icon, check whether a graphic is liked, or confirm that information has been appropriately categorized on a webpage. This approach is often called "quick and dirty" because it is meant to be done in a short space of time. Getting this kind of feedback is an essential ingredient of successful design.
As discussed in earlier lectures, any involvement with users will be highly informative and you can learn a lot early in design by observing what people do and talking to them informally. The data collected is usually descriptive and informal and it is fed back into

© Copyright Virtual University of Pakistan

the design process as verbal or written notes, sketches and anecdotes, etc. Another source comes from consultants, who use their knowledge of user behavior, the market place and technical know-how, to review software quickly and provide suggestions for improvement. It is an approach that has become particularly popular in web design where the emphasis is usually on short timescales.

Usability testing is a controlled method where real users perform tasks to measure how easily and effectively they can use a system, while their actions and feedback are recorded and analyzed.

Usability testing

Usability testing was the dominant approach in the 1980s (Whiteside et al., 1998), and remains important, although, as you will see, field studies and heuristic evaluations have grown in prominence. Usability testing involves measuring typical users' performance on carefully prepared tasks that are typical of those for which the system was designed. Users' performance is generally measured in terms of number of errors and time to complete the task. As the users perform these tasks, they are watched and recorded on video and by logging their interactions with software. This observational data is used to calculate performance times, identify errors, and help explain why the users did what they did. User satisfaction questionnaires and interviews are also used to elicit users' opinions.
The defining characteristic of usability testing is that it is strongly controlled by the evaluator (Mayhew. 1999). There is no mistaking that the evaluator is in charge! Typically tests take place in laboratory-like conditions that are controlled. Casual visitors are not allowed and telephone calls are stopped, and there is no possibility of talking to colleagues, checking email, or doing any of the other tasks that most of us rapidly switch among in our normal lives. Everything that the participant does is recorded—every key press, comment, pause, expression, etc., so that it can be used as data.
Quantifying users' performance is a dominant theme in usability testing. However, unlike research experiments, variables are not manipulated and the typical number of participants is too small for much statistical analysis. User satisfaction data from questionnaires tends to be categorized and average ratings are presented. Sometimes video or anecdotal evidence is also included to illustrate problems that users encounter. Some evaluators then summarize this data in a usability specification so that developers can use it to test future prototypes or versions of the product against it. Optimal performance levels and minimal levels of acceptance are often specified and current levels noted. Changes in the design can then be agreed and engineered—hence the term "usability engineering.

Field studies

The distinguishing feature of field studies is that they are done in natural settings with the aim of increasing understanding about what users do naturally and how technology impacts them. In product design, field studies can be used to (1) help identify opportunities for new technology; (2) determine requirements for design: (3) facilitate the introduction of technology: and (4) evaluate technology (Bly. 1997).
We introduced qualitative techniques such as interviews, observation, participant observation, and ethnography that are used in field studies. The exact choice of techniques is often influenced by the theory used to analyze the data. The data takes the form of events and conversations that are recorded as notes, or by audio or video recording, and later analyzed using a variety of analysis techniques such as content, discourse, and conversational analysis. These techniques vary considerably. In content analysis, for example, the data is analyzed into content categories, whereas in discourse analysis the use of words and phrases is examined. Artifacts are also collected. In fact,

© Copyright Virtual University of Pakistan

anything that helps to show what people do in their natural contexts can be regarded as data.
In this lecture we distinguish between two overall approaches to field studies. The first involves observing explicitly and recording what is happening, as an outsider looking on. Qualitative techniques are used to collect the data, which may then he analyzed qualitatively or quantitatively. For example, the number of times a particular event is observed may be presented in a bar graph with means and standard deviations.

In some field studies the evaluator may be an insider or even a participant. Ethnography is a particular type of insider evaluation in which the aim is to explore the details of what happens in a particular social setting. “In the context of human computer interaction, ethnography is a means of studying work (or other activities) in order to inform the design of information systems and understand aspects of their use” (Shapiro, 1995, p. 8).

Predictive evaluation

In predictive evaluations experts apply their knowledge of typical users, often guided by heuristics, to predict usability problems. Another approach involves theoretically based models. The key feature of predictive evaluation is that users need *not* be present, which makes the process quick, relatively inexpensive, and thus attractive to companies; but it has limitations.
In recent years heuristic evaluation in which experts review the software product guided by tried and tested heuristics has become popular (Nielsen and Mack, 1994). Usability guidelines (e.g., always provide clearly marked exits) were designed primarily for evaluating screen-based products (e.g. form fill-ins, library catalogs, etc.). With the advent of a range of new interactive products (e.g., the web, mobiles, collaborative technologies), this original set of heuristics has been found insufficient. While some are still applicable (e.g., speak the users' language), others are inappropriate. New sets of heuristics are also needed that are aimed at evaluating different classes of interactive products. In particular, specific heuristics are needed that are tailored to evaluating web-based products, mobile devices, collaborative technologies, computerized toys, etc. These should be based on a combination of usability and user experience goals, new research findings and market research. Care
is needed in using sets of heuristics. Designers are sometimes led astray by findings from heuristic evaluations that turn out not to be as accurate as they at first seemed.
Table bellow summarizes the key aspects of each evaluation paradigm for the following issues:

- the role of users
- who controls the process and the relationship between evaluators and users during the evaluation
- the location of the evaluation
- when the evaluation is most useful
- the type of data collected and how it is analyzed
- how the evaluation findings are fed back into the design process
- the philosophy and theory that underlies the evaluation paradigms.

© Copyright Virtual University of Pakistan

Evaluation paradigms	"Quick and dirty"	Usability testing	Field studies	Predictive
Role of users	Natural behavior.	To carry out set tasks.	Natural behavior.	Users generally not involved.
Who controls	E valuators take minimum control.	Evaluators strongly in control.	Evaluators try to develop relationships with users.	Expert evaluators.
Location	Natural environment or laboratory	Laboratory.	Natural environment.	Laboratory-oriented but often happens on customer's premises.
When used	Any time you want to get feedback about a design quickly. Techniques from other evaluation paradigms can be Used e.g. experts review soft ware.	With a prototype or product.	Most often used early in design to check that users' needs are being met or to assess problems or design opportunities.	Expert reviews (often done by consultants) with a prototype, but can occur at any time. Models are used to assess specific aspects of a potential design.
Type of data	Usually qualitative, informal descriptions	Quantitative. Sometimes statistically validated. Users' opinions collected by questionnaire or interview.	Qualitative descriptions often accompanied with sketches. Scenarios quotes, other artifacts.	List of problems from expert reviews. Quantitative figures from model, e.g., how long it takes to perform a task using two designs.

© Copyright Virtual University of Pakistan

Fed back into design by..	Sketches, quotes, descriptive report.	Report of performance measures, errors etc. Findings provide a benchmark for future versions.	Descriptions that include quotes, Sketches, anecdotes, and sometimes time logs.	Reviewers provide a list of problems, often with suggested solutions. Times calculated from models are given to designers.
Philosophy	User-centered, highly practical approach	Applied approach based on experimentation. i.e., usability engineering.	May be objective observation or ethnographic.	Practical heuristics and practitioner expertise underpin expert reviews. Theory underpins models

Techniques

There are many evaluation techniques and they can be categorized in various ways, but in this lecture we will examine techniques for:

- observing users
- asking users their opinions
- asking experts their opinions
- testing users" performance
- modeling users' task performance to predict the efficacy of a user interface

The brief descriptions below offer an *overview* of each category. Be aware that some techniques are used in different ways in different evaluation paradigms.

Observing users

Observation techniques help to identify needs leading to new types of products and help to evaluate prototypes. Notes, audio, video, and interaction logs are well-known ways of recording observations and each has benefits and drawbacks. Obvious challenges for evaluators are how to observe without disturbing the people being observed and how to analyze the data, particularly when large quantities of video data are collected or when several different types must be integrated to tell the story (e.g., notes, pictures, and sketches from observers).

Asking users

Asking users what they think of a product—whether it does what they want; whether they like it; whether the aesthetic design appeals; whether they had problems using it; whether they want to use it again—is an obvious way of getting feedback. Inter views and questionnaires are the main techniques for doing this. The questions asked can be unstructured or tightly structured. They can be asked of a few people or of hundreds.

© Copyright Virtual University of Pakistan

Interview and questionnaire techniques are also being developed for use with email and the web.

Asking experts

Software inspections and reviews are long established techniques for evaluating software code and structure. During the 1980s versions of similar techniques were developed for evaluating usability. Guided by heuristics, experts step through tasks role-playing typical users and identify problems. Developers like this approach he-cause it is usually relatively inexpensive and quick to perform compared with laboratory and field evaluations that involve users. In addition, experts frequently suggest solutions to problems

User testing

Measuring user performance to compare two or more designs has been the bedrock of usability testing. As we said earlier when discussing usability testing, these tests are usually conducted in controlled settings and involve typical users performing typical. well-defined tasks. Data is collected so that performance can be analyzed. Generally the time taken to complete a task, the number of errors made, and the navigation path through the product are recorded. Descriptive statistical measures such as means and standard deviations are commonly used to report the results.

Modeling users' task performance

There have been various attempts to model human-computer interaction so as to predict the efficiency and problems associated with different designs at an early stage without building elaborate prototypes. These techniques are successful for systems with limited functionality such as telephone systems. GOMS and the keystroke model are the best known techniques.

© Copyright Virtual University of Pakistan

Lecture 30.

Evaluation – Part II

Learning Goals

The aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the DECIDE evaluation framework

30.1 DECIDE: A framework to guide evaluation

Well-planned evaluations are driven by clear goals and appropriate questions (Basili et al., 1994). To guide our evaluations we use the DECIDE framework, which provides the following checklist to help novice evaluators:

1. Determine the overall *goals* that the evaluation addresses.
2. Explore the specific *questions* to be answered.
3. Choose the *evaluation paradigm* and *techniques* to answer the questions.
4. Identify the *practical issues* that must be addressed, such as selecting participants.
5. Decide how to deal with the *ethical issues.*
6. Evaluate, interpret, and present the *data.*

Determine the goals

Determining the goals means clearly defining why the evaluation is being done and what it aims to achieve, such as understanding user needs or improving the interface.

What are the high-level goals of the evaluation? Who wants it and why? An evaluation to help clarify user needs has different goals from an evaluation to determine the best metaphor for a conceptual design, or to fine-tune an interface, or to examine how technology changes working practices, or to inform how the next version of a product should be changed.

Goals should guide an evaluation, so determining what these goals are is the first step in planning an evaluation. For example, we can restate the general goal statements just mentioned more clearly as:

- Check that the evaluators have understood the users' needs.
- Identify the metaphor on which to base the design.
- Check to ensure that the final interface is consistent.
- Investigate the degree to which technology influences working practices.
- Identify how the interface of an existing product could be engineered to improve its usability.

These goals influence the evaluation approach, that is, which evaluation paradigm guides the study. For example, engineering a user interface involves a quantitative engineering style of working in which measurements are used to judge the quality of the interface. Hence usability testing would be appropriate. Exploring how children talk together in order to see if an innovative new groupware product would help them to be more engaged would probably be better informed by a field study.

© Copyright Virtual University of Pakistan

Explore the questions

Exploring the questions means breaking down the evaluation goals into specific questions that help uncover the exact problems and user concerns.

In order to make goals operational, questions that must be answered to satisfy them have to be identified. For example, the goal of finding out why many customers prefer

to purchase paper airline tickets over the counter rather than e-tickets can he broken down into a number of relevant questions for investigation. What are customers' attitudes to these new tickets? Perhaps they don't trust the system and are not sure that they will actually get on the flight without a ticket in their hand. Do customers have adequate access to computers to make bookings? Are they concerned about security? Does this electronic system have a bad reputation? Is the user interface to the ticketing system so poor that they can't use it? Maybe very few people managed to complete the transaction.
Questions can be broken down into very specific sub-questions to make the evaluation even more specific. For example, what does it mean to ask, "Is the user interface poor?": Is the system difficult to navigate? Is the terminology confusing because it is inconsistent? Is response time too slow? Is the feedback confusing or maybe insufficient? Sub-questions can, in turn, be further decomposed into even finer-grained questions, and so on.

Choose the evaluation paradigm and techniques

This means selecting the best evaluation method and tools based on your goals, questions, resources, time, and ethical considerations.

Having identified the goals and main questions, the next step is to choose the evaluation paradigm and techniques. As discussed in the previous section, the evaluation paradigm determines the kinds of techniques that are used. Practical and ethical issues (discussed next) must also be considered and trade-offs made. For example, what seems to be the most appropriate set of techniques may be too expensive, or may take too long, or may require equipment or expertise that is not available, so compromises are needed.

Identify the practical issues

There are many practical issues to consider when doing any kind of evaluation and it is important to identify them *before* starting. Some issues that should be considered include users, facilities and equipment, schedules and budgets, and evaluators' expertise. Depending on the availability of resources, compromises may involve adapting or substituting techniques.

Users

It goes without saying that a key aspect of an evaluation is involving *appropriate* users. For laboratory studies, users must be found and screened to ensure that they represent the user population to which the product is targeted. For example, usability tests often need to involve users with a particular level of experience e.g., novices or experts, or users with a range of expertise. The number of men and women within a particular age range, cultural diversity, educational experience, and personality differences may also need to be taken into account, depending on the kind of product being evaluated. In usability tests participants are typically screened to ensure that they meet some predetermined characteristic. For example, they might be tested to ensure that they have attained a certain skill level or fall within a particular demographic range. Questionnaire surveys require large numbers of participants so ways of identifying and reaching a representative sample of participants are needed. For field studies to be successful, an appropriate and accessible site must be found where the evaluator can work with the users in their natural setting.

© Copyright Virtual University of Pakistan

Another issue to consider is how the users will be involved. The tasks used in a laboratory study should be representative of those for which the product is de signed. However, there are no written rules about the length of time that a user should be expected to spend on an evaluation task. Ten minutes is too short for most tasks and two hours is a long time, but what is reasonable? Task times will vary according to the type of evaluation, but when tasks go on for more than 20 minutes, consider offering breaks. It is accepted that people using computers should stop, move around and change their position regularly after every 20 minutes spent at the keyboard to avoid repetitive strain injury. Evaluators also need to put users at ease so they are not anxious and will perform normally. Even when users are paid to participate, it is important to treat them courteously. At no time should users be treated condescendingly or made to feel uncomfortable when they make mistakes. Greeting users, explaining that it is the system that is being tested and not them, and planning an activity to familiarize them with the system before starting the task all help to put users at ease.

Facilities and equipment

There are many practical issues concerned with using equipment in an evaluation For example, when using video you need to think about how you will do the recording: how many cameras and where do you put them? Some people are disturbed by having a camera pointed at them and will not perform normally, so how can you avoid making them feel uncomfortable? Spare film and batteries may also be needed.

Schedule and budget constraints

Time and budget constraints are important considerations to keep in mind. It might seem ideal to have 20 users test your interface, but if you need to pay them, then it could get costly. Planning evaluations that can be completed on schedule is also important, particularly in commercial settings. There is never enough time to do evaluations as you would ideally like, so you have to compromise and plan to do a good job with the resources and time available.

Expertise

Does the evaluation team have the expertise needed to do the evaluation? For example, if no one has used models to evaluate systems before, then basing an evaluation on this approach is not sensible. It is no use planning to use experts to review an interface if none are available. Similarly, running usability tests requires expertise. Analyzing video can take many hours, so someone with appropriate expertise and equipment must be available to do it. If statistics are to be used, then a statistician should be consulted before starting the evaluation and then again later for analysis, if appropriate.

Decide how to deal with the ethical issues

The Association for Computing Machinery (ACM) and many other professional organizations provide ethical codes that they expect their members to uphold, particularly if their activities involve other human beings. For example. people's privacy should be protected, which means that their name should not be associated with data collected about them or disclosed in written reports (unless they give permission). Personal records containing details about health, employment, education, financial status, and where participants live should be confidential. Similarly, it

© Copyright Virtual University of Pakistan

should not be possible to identify individuals from comments written in reports For example, if a focus group involves nine men and one woman, the pronoun “she” should not be used in the report because it will be obvious to whom it refers

Most professional societies, universities, government and other research offices require researchers to provide information about activities in which human participants will be involved. This documentation is reviewed by a panel and the researchers are notified whether their plan of work, particularly the details about how human participants will be treated, is acceptable.

People give their time and their trust when they agree to participate in an evaluation study and both should be respected. But what does it mean to be respectful to users? What should participants be told about the evaluation? What are participants’ rights? Many institutions and project managers require participants to read and sign an informed consent. This form explains the aim of the tests or research and promises participants that their personal details and performance will not be made public and will be used only for the purpose stated. It is an agreement between the evaluator and the evaluation participants that helps to confirm the professional relationship that exists between them. *I*f your university or organization does not provide such a form it is advisable to develop one, partly to protect yourself in the unhappy event of litigation and partly because the act of constructing it will remind you what you should consider.

The following guidelines will help ensure that evaluations are done ethically and that adequate steps to protect users' rights have been taken.

- Tell participants the goals of the study and exactly what they should expect if they participate. The information given to them should include outlining the process, the approximate amount of time the study will take, the kind of data that will be collected, and how that data will be analyzed. The form of the final report should be described and, if possible, a copy offered to them. Any payment offered should also be clearly stated.
- Be sure to explain that demographic, financial, health, or other sensitive information that users disclose or is discovered from the tests is confidential. A coding system should be used to record each user and, if a user must be identified for a follow-up interview, the code and the person's demographic details should be stored separately from the data. Anonymity should also be promised if audio and video are used.
- Make sure users know that they are free to stop the evaluation at any time if they feel uncomfortable with the procedure.
- Pay users when possible because this creates a formal relationship in which mutual commitment and responsibility are expected.
- Avoid including quotes or descriptions that inadvertently reveal a person's identity, as in the example mentioned above, of avoiding use of the pronoun "she" in the focus group. If quotes need to be reported, e.g., to justify conclusions, then it is convention to replace words that would reveal the source with representative words, in square brackets. Ask users' permission in advance to quote them, promise them anonymity, and offer to show them a copy of the report before it is distributed.

The general rule to remember when doing evaluations is do unto others only what you would not mind being done to you.

© Copyright Virtual University of Pakistan

The recent explosion in Internet and web usage has resulted in more research on how people use these technologies and their effects on everyday life. Consequently, there are many projects in which developers and researchers are logging users' interactions, analyzing web traffic, or examining conversations in chat rooms, bulletin boards, or on email. Unlike most previous evaluations in human-computer interaction, these studies can be done without users knowing that they are being studied. This raises ethical concerns, chief among which are issues of privacy, confidentiality, informed consent, and appropriation of others' personal stories (Sharf, 1999). People often say things online that they would not say face to face. Further more, many people are unaware that personal information they share online can be read by someone with technical know-how years later, even after they have deleted it from their personal mailbox (Erickson et aL 1999).

Evaluate, interpret, and present the data

Choosing the evaluation paradigm and techniques to answer the questions that satisfy the evaluation goal is an important step. So is identifying the practical and ethical issues to be resolved. However, decisions are also needed about what data to collect, how to analyze it, and how to present the findings to the development team. To a great extent the technique used determines the type of data collected, but there are still some choices. For example, should the data be treated statistically? If qualitative data is collected, how should it be analyzed and represented? Some general questions also need to be asked (Preece et al., 1994): Is the technique reliable? Will the approach measure what is intended, i.e., what is its validity? Are biases creeping in that will distort the results? Are the results generalizable, i.e., what is their scope? Is the evaluation ecologically valid or is the fundamental nature of the process being changed by studying it?

Reliability

The reliability or consistency of a technique is how well it produces the *same* results on separate occasions under the *same* circumstances. Different evaluation processes have different degrees of reliability. For example, a carefully controlled experiment will have high reliability. Another evaluator or researcher who follows exactly the same procedure should get similar results. In contrast, an informal, unstructured interview will have low reliability: it would be difficult if not impossible to repeat exactly the same discussion.

Validity

Validity means checking if you're testing the right thing in the right way.

Validity is concerned with whether the evaluation technique measures what it is supposed to measure. This encompasses both the technique itself and the way it is performed. If for example, the goal of an evaluation is to find out how users use a new product in their homes, then it is not appropriate to plan a laboratory experiment. An ethnographic study in users' homes would be more appropriate. If the goal is to find average performance times for completing a task, then counting only the number of user errors would be invalid.

Biases

Biases are mistakes in evaluation that happen when personal opinions or actions affect the results.

Bias occurs when the results are distorted. For example, expert evaluators performing a heuristic evaluation may be much more sensitive to certain kinds of design flaws than others. Evaluators collecting observational data may consistently fail to notice certain types of behavior because they do not deem them important.

Put another way, they may selectively gather data that they think is important. Interviewers may unconsciously influence responses from interviewees by their tone of

© Copyright Virtual University of Pakistan

voice, their facial expressions, or the way questions are phrased, so it is important to be sensitive to the possibility of biases.

Scope

The scope of an evaluation study refers to how much its findings can be generalized. For example, some modeling techniques, like the keystroke model, have a narrow, precise scope. The model predicts expert, error-free behavior so, for example, the results cannot be used to describe novices learning to use the system.

Ecological validity

Ecological validity concerns how the environment in which an evaluation is conducted influences or even distorts the results. For example, laboratory experiments are strongly controlled and are quite different from workplace, home, or leisure environments. Laboratory experiments therefore have low ecological validity because the results are unlikely to represent what happens in the real world. In contrast, ethnographic studies do not impact the environment, so they have high ecological validity.

Ecological validity is also affected when participants are aware of being studied. This is sometimes called the *Hawthorne effect* after a series of experiments at the Western Electric Company's Hawthorne factory in the US in the 1920s and 1930s. The studies investigated changes in length of working day, heating, lighting etc., but eventually it was discovered that the workers were reacting positively to being given special treatment rather than just to the experimental conditions

© Copyright Virtual University of Pakistan

Lecture 31.

Evaluation – Part VII

Learning Goals

The aim of this lecture is to understand how to perform evaluation through usability testing.

What is Usability Testing?

While there can be wide variations in where and how you conduct a usability test, every usability test shares these five characteristics:

1. The primary goal is to improve the usability of a product. For each test, you also have more specific goals and concerns that you articulate when planning the test.
2. The participants represent real users.
3. The participants do real tasks.
4. You observe and record what participants do and say.
5. You analyze the data, diagnose the real problems, and recommend changes to fix those problems.

The Goal is to improve the Usability of a Product

The primary goal of a usability test is to improve the usability of the product that is being tested. Another goal, as we will discuss in detail later, is to improve the process by which products are designed and developed, so that you avoid having the same problems again in other products.

This characteristic distinguishes a usability test from a research study, in which the goal is to investigate the existence of some phenomenon. Although the same facility might be used for both, they have different purposes. This characteristic also distinguishes a usability test from a quality assurance or function test, which has a goal of assessing whether the product works according to its specifications.

Within the general goal of improving the product, you wilI have more specific goals and concerns that differ from one test to another.

You might be particularly concerned about how easy it is for users to navigate through the menus. You could test that concern before coding the product, by creating an interactive prototype of the menus, or by giving users paper versions of each screen.

You might be particularly concerned about whether the interface that you have developed for novice users will also be easy for and acceptable to experienced users.

For one test, you might be concerned about how easily the customer representatives who do installations will be able to install the product. For another test, you might be concerned about how easily the client's nontechnical staff will be able to operate and maintain the product.

These more specific goals and concerns help determine which users are appropriate participants for each test and which tasks are appropriate to have them do during the test.

The Participants Represent Real Users

© Copyright Virtual University of Pakistan

The people who come to test the product must be members of the group of people who now use or who will use the product. A test that uses programmers when the product is intended for legal secretaries is not a usability test.
The quality assurance people who conduct function tests may also find usability problems, and the problems they find should not be ignored, but they are not conducting a usability test. They are not real users-unless it is a product about function testing. They are acting more like expert reviewers.
If the participants are more experienced than actual users, you may miss problems that will cause the product to fail in the marketplace. If the participants are less experienced than actual users, you may be led to make changes that aren't improvements for the real users.

The Participants Do Real Tasks
The tasks that you have users do in the test must be ones that they will do with the product on their jobs or in their homes. This means that you have to understand users' jobs and the tasks for which this product is relevant.
In many usability tests, particularly of functionally rich and complex software products, you can only test some of the many tasks that users will be able to do with the product. In addition to being realistic and relevant for users, the tasks that you include in a test should relate to your goals and concerns and have a high probability of uncovering a usability problem.

Observe and Record What the Participants Do and Say
In a usability test, you usually have several people come, one at a time, to work with the product. You observe the participant, recording both performance and comments.
You also ask the participant for opinions about the product. A usability test includes both times when participants are doing tasks with the product and times when they are filling out questionnaires about the product.

Observing and recording individual participant's behaviors distinguishes a usability test from focus groups, surveys, and beta testing.
A typical focus group is a discussion among 8 to 10 real users, led by a professional moderator. Focus groups provide information about users' opinions, attitudes, preferences, and their self-report about their performance, but focus groups do not usually let you see how users actually behave with the product.
Surveys, by telephone or mail, let you collect information about users' opinions, attitudes, preferences, and their self-report of behavior, but you cannot use a survey to observe and record what users actually do with a product.
A typical beta test (field test, clinical trial, user acceptance test) is an early release of a product to a few users. A beta test has ecological validity, that is, real people are using the product in real environments to do real tasks. However, beta testing seldom yields any useful information about usability. Most companies have found beta testing to be too little, too unsystematic, and much too late to be the primary test of usability.

Analyze the Data, Diagnose the Real Problems, and Recommend Changes to Fix Those Problems
Collecting the data is necessary, but not sufficient, for a usability test. After the test itself, you still need to analyze the data. You consider the quantitative and qualitative data from the participants together with your own observations and users' comments. You use all of

© Copyright Virtual University of Pakistan

that to diagnose and document the product's usability problems and to recommend solutions to those problems.

The Results Are Used to Change the Product - and the Process

We would also add another point. It may not be part of the definition of the usability test itself, as the previous five points were, but it is crucial, nonetheless.

A usability test is not successful if it is used only to mark off a milestone on the development schedule. A usability test is successful only if it helps to improve the product that was tested and the process by which it was developed.

What Is Not Required for a Usability Test?

Our definition leaves out some features you may have been expecting to see, such as:

- a laboratory with one-way mirror
- data-logging software
- videotape
- a formal test report

Each of these is useful, but not necessary, for a successful usability test. For example, a memorandum of findings and recommendations or a meeting about the test results, rather than a formal test report, may be appropriate in your situation.

Each of these features has advantages in usability testing that we discuss in detail later, but none is an absolute requirement. Throughout the book, we discuss methods that you can use when you have only a shoestring budget, limited staff, and limited testing equipment.

When is a Usability Test Appropriate?

Nothing in our definition of a usability test limits it to a single, summative test at the end of a project. The five points in our definition are relevant no matter where you are in the design and development process. They apply to both informal and formal testing. When testing a prototype, you may have fewer participants and fewer tasks, take fewer measures, and have a less formal reporting procedure than in a later test, but the critical factors we outline here and the general process we describe in this book still apply. Usability testing is appropriate iteratively from predesign (test a similar product or earlier version), through early design (test prototypes), and throughout development (test different aspects, retest changes).

Questions that Remain in Defining Usability Testing

We recognize that our definition of usability testing still has some fuzzy edges.

- Would a test with only one participant be called a usability test? Probably not. You probably need at least two or three people representing a subgroup of users to feel comfortable that you are not seeing idiosyncratic behavior.
- Would a test in which there were no quantitative measures qualify as a usability test? Probably not. To substantiate the problems that you report, we assume that you will take at least some basic measures, such as number of participants who had the problem, or number of wrong choices, or time to complete a task. The actual measures will depend on your specific concerns and the stage of design or

© Copyright Virtual University of Pakistan

development at which you are testing. The measures could come from observations, from recording with a data-logging program, or from a review of the videotape after the test. The issue is not which measures or how you collect them, but whether you need to have some quantitative data to have a usability test.

Usability testing is still a relatively new development; its definition is still emerging. You may have other questions about what counts as a usability test. Our discussion of usability testing and of other usability engineering methods, in this chapter and the next three chapters, may help clarify your own thinking about how to define usability testing.

Testing Applies to All Types of Products

If you read the literature on usability testing, you might think that it is only about testing software for personal computers. Not so. Usability testing works for all types of products. In the last several years, we've been involved in usability testing of all these products:

Consumer products
Regular TVs
High-definition
TVs
VCRs
Cordless telephones
Telephone/answering machines
Business telephones

Medical products
Bedside terminal Anesthesiologist's workstation
Patient monitors Blood gas analyzer
Integrated communication system for wards
Nurse's workstation for intensive care units

Engineering devices
Digital oscilloscope

Network protocol analyzer (for maintaining computer networks)
Application software for microcomputers, minicomputers, and mainframes
Electronic mail Database management software
Spreadsheets Time management software
Compilers and debuggers for programming languages Operating system software

Other
Voice response systems (menus on the telephone)
Automobile navigation systems (in-car information about how to get where you want to go)

The procedures for the test may vary somewhat depending on what you are testing and the questions you are asking. We give you hints and tips, where appropriate, on special concerns when you are focusing the testing on hardware or documentation; but, in general, we don't find that you need to change the approach much at all.

Most of the examples in this book are about testing some type of hardware or software and the documentation that goes with it. In some cases, the hardware used to be just a

© Copyright Virtual University of Pakistan

machine and is now a special purpose computer. For usability testing, however, the product doesn't even have to involve any hardware or software. You can use the techniques in this book to develop usable

. Application or reporting forms
. Instructions for non-computer products, like bicycles. Interviewing techniques
. Non-automated procedures
. Questionnaires

Testing All Types of Interfaces

Any product that people have to use, whether it is computer-based or not, has a user interface. Norman in his marvelous book, The Design of Everyday Things (1988) points out problems with doors, showers, light switches, coffee pots, and many other objects that we come into contact with in our daily lives. With creativity, you can plan a test of any type of interface.

Consider an elevator. The buttons in the elevator are an interface- the way that you, the user, talk to the computer that now drives the machine. Have you ever been frustrated by the way the buttons in an elevator are arranged? Do you search for the one you want? Do you press the wrong one by mistake?

You might ask: How could you test the interface to an elevator in a usability laboratory? How could the developers find the problems with an elevator interface before building the elevator-at which point it would be too expensive to change?

In fact, an elevator interface could be tested before it is built. You could create a simulation of the proposed control panel on a touch screen computer (a prototype). You could even program the computer to make the alarm sound and to make the doors seem to open and close, based on which buttons users touch. Then you could bring in users one at a time, give them realistic situations, and have them use the touch screen as they would the panel in the elevator.

Testing All Parts of the Product

Depending on where in the development process you are and what you are particularly concerned about, you may want to focus the usability test on a specific part of the product, such as

. Installing hardware
. Operating hardware
. Cleaning and maintaining hardware
. Understanding messages about the hardware
. Installing software
. Navigating through menus
. Filling out fields
. Recovering from errors
. Learning from online or printed tutorials
. Finding and following instructions in a user's guide. Finding and following instructions in the on line help

© Copyright Virtual University of Pakistan

Testing Different Aspects of the Documentation

When you include documentation in the test, you have to decide if you are more interested in whether users go to the documentation or in how well the documentation works for them when they do go to it. It is difficult to get answers to both of those concerns at the same time.

If you want to find out how much people learn from a tutorial when they use it, you can set up a test in which you ask people to go through the tutorial. Your test participants will do as you ask, and you will get useful information about the design, content, organization, and language of the tutorial.

You will, however, not have any indication of whether anyone will actually open the tutorial when they get the product. To test that, you have to set up your test differently.

Instead of instructing people to use the tutorial, you have to give them tasks and let them know the tutorial is available. In this second type of test, you will find out which types of users are likely to try the tutorial, but if few participants use it, you won't get much useful information for revising the tutorial.

Giving people instructions that encourage them to use the manual or tutorial may be unrealistic in terms of what happens in the world outside the test laboratory, but it is necessary if your concern is the usability of the documentation. At some point in the process of developing the product, you should be testing the usability of the various types of documentation that users will get with the product.

At other points, however, you should be testing the usability of the product in the situation in which most people will receive it. Here's an example:

A major company was planning to put a new software product on its internal network. The product has online help and a printed manual, but, in reality, few users will get a copy of the manual.

The company planned to maintain a help desk, and a major concern for the usability test was that if people don't get the manual, they would have to use the online help, call the help desk, or ask a co-worker. The company wanted to keep calls to the help desk to a minimum, and the testers knew that when one worker asks another for help, two people are being unproductive for the company.

When they tested the product, therefore, this test team did not include the manual. Participants were told that the product includes online help, and they were given the phone number of the help desk to call if they were really stuck. The test team focused on where people got stuck, how helpful the online help was, and at what point’s people called the help desk.

This test gave the product team a lot of information to improve the interface and the online help to satisfy the concern that drove the test. However, this test yielded no information to improve the printed manual. That would require a different test.

Testing with Different Techniques

In most usability tests, you have one participant at a time working with the product. You usually leave that person alone and observe from a corner of the room or from behind a one-way mirror. You intervene only when the person "calls the help desk," which you record as a need for assistance.

© Copyright Virtual University of Pakistan

You do it this way because you want to simulate what will happen when individual. users get the products in their offices or homes. They'll be working on their own, and you won't be right there in their rooms to help them.
Sometimes, however, you may want to change these techniques. Two ideas that many teams have found useful are:
. Co-discovery, having two participants works together
. Active intervention, taking a more active role in the test

Co-discovery
Co-discovery is a technique in which you have two participants work together to perform the tasks (Kennedy, 1989). You encourage the participants to talk to each other as they work.
Talking to another person is more natural than thinking out loud alone. Thus, co-discovery tests often yield more information about what the users are thinking and what strategies they are using to solve their problems than you get by asking individual participants to think out loud.
Hackman and Biers (1992) have investigated this technique. They confirmed that co-discovery participants make useful comments that provide insight into the design.

They also found that having two people work together does not distort other results. Participants who worked together did not differ in their performance or preferences from participants who worked alone.
Co-discovery is more expensive than single participant testing, because you have to pay two people for each session. In addition, it may be more difficult to watch two people working with each other and the product than to watch just one person at a time. Co-discovery may be used anytime you conduct a usability test, but it is especially useful early in design because of the insights that the participants provide as they talk with each other.

Active Intervention
Active intervention is a technique in which a member of the test team sits in the room with the participant and actively probes the participant's understanding of whatever is being tested. For example, you might ask participants to explain what they would do next and why as they work through a task. When they choose a particular menu option, you might ask them to describe their understanding of the menu structure at that moment. By asking probing questions throughout the test, rather than in one interview at the end, you can get insights into participants' evolving mental model of the product.
You can get a better understanding of problems that participants are having than by just watching them and hoping they'll think out loud.
Active intervention is particularly useful early in design. It is an excellent technique to use with prototypes, because it provides a wealth of diagnostic information. It is not the technique to use, however, if your primary concern is to measure time to complete tasks or to find out how often users will call the help desk.
To do a useful active intervention test, you have to define your goals and concerns, plan the questions you will use as probes, and be careful not to bias participants by asking leading questions.

© Copyright Virtual University of Pakistan

Additional Benefits of Usability Testing
Usability testing contributes to all the benefits of focusing on usability that we gave in Chapter 1. In addition, the process of usability testing has two specific benefits that may not be as strong or obvious from other usability techniques. Usability testing helps
. Change people's attitudes about users

. Change the design and development process

Changing People's Attitudes about Users
Watching users is both inspiring and humbling. Even after watching hundreds of people participate in usability tests, we are still amazed at the insights they give us about the assumptions we make.

When designers, developers, writers, and managers attend a usability test or watch videotapes from a usability test for the first time, there is often a dramatic transformation in the way that they view users and usability issues. Watching just a few people struggle with a product has a much greater impact on attitudes than many hours of discussion about the importance of usability or of understanding users.

After an initial refusal to believe that the users in the test really do represent the people for whom the product is meant, many observers become instant converts to usability. They become interested not only in changing this product, but in improving all future products, and in bringing this and other products back for more testing.

Changing the Design and Development Process
In addition to helping to improve a specific product, usability testing can help improve the process that an organization uses to design and develop products (Dumas, 1989). The specific instances that you see in a usability test are most often symptoms of broader and deeper global problems with both the product and the process.

Comparing Usability Testing to Beta Testing
Despite the surge in interest in usability testing, many companies still do not think about usability until the product is almost ready to be released. Their usability approach is to give some customers an early-release (almost ready) version of the product and wait for feedback. Depending on the industry and situation, these early release trials may be called beta testing, field testing, clinical trials, or user acceptance testing.
In beta testing, real users do real tasks in their real environments. However, many companies find that they get very little feedback from beta testers, and beta testing seldom yields useful information about usability problems for these reasons:

. The beta test site does not even have to use the product.

. The feedback is unsystematic. Users may report-after the fact-what they remember and choose to report. They may get so busy that they forget to report even when things go wrong.

. In most cases, no one observes the beta test users and records their behavior. Because users are focused on doing their work, not on testing the product, they may not be able to recall the actions they took that resulted in the problems. In a usability test, you get to see

© Copyright Virtual University of Pakistan

the actions, hear the users talk as they do the actions, and record the actions on videotape so that you can go back later and review them, if you aren't sure what the user did.

. In a beta test, you do not choose the tasks. The tasks that get tested are whatever users happen to do in the time they are working with the product. A situation that you are concerned about may not arise. Even if it does arise, you may not hear about it. In a usability test, you choose the tasks that participants do with the product. That way, you can be sure that you get information about aspects of the product that relate to your goals and concerns. That way, you also get comparable data across participants.

If beta testers do try the product and have major problems that keep them from completing their work, they may report those problems. The unwanted by-product of that situation, however, may be embarrassment at having released a product with major problems, even to beta testers.

Even though beta testers know that they are working with an unfinished and possibly buggy product, they may be using it to do real work where problems may have serious consequences. They want to do their work easily and effectively. Your company's reputation and sales may suffer if beta testers find the product frustrating to use. A bad experience when beta testing your product may make the beta testers less willing to buy the product and less willing to consider other products from your company.

You can improve the chances of getting useful information from beta test sites. Some companies include observations and interviews with beta testing, going out to visit beta test sites after people have been working with the product for a while. Another idea would be to give tape recorders to selected people at beta test sites and ask them to talk on tape while they use the product or to record observations and problems as they occur.

Even these techniques, however, won't overcome the most significant disadvantage of beta testing-that it comes too late in the process. Beta testing typically takes place only very close to the end of development, with a fully coded product. Critical functional bugs may get fixed after beta testing, but time and money generally mean that usability problems can't be addressed.

Usability testing, unlike beta testing, can be done throughout the design and development process. You can observe and record users as they work with prototypes and partially developed products. People are more tolerant of the fact that the product is still under development when they come to a usability test than when they beta test it. If you follow the usability engineering approach, you can do usability testing early enough to change the product-and retest the changes.

© Copyright Virtual University of Pakistan

Lecture 32.

Evaluation IV

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the significance of navigation

People won't use your Web site if they can't find their way around it.

You know this from your own experience as a Web user. If you go to a site and can't find what you're looking for or figure out how the site is organized, you're not likely to stay long—or come back. So how do you create the proverbial "clear, simple, and consistent" navigation?

32.1 Scene from a mall

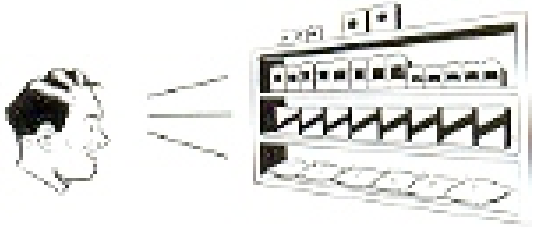
Picture this: It's Saturday afternoon and you're headed for the mall to buy a chainsaw.
As you walk through the door at Sears, you're thinking, "Hmmm. Where do they keep chainsaws?" As soon as you're inside, you start looking at the department names, high up on the walls. (They're big enough that you can read them from all the way across the store.)



"Hmmm," you think, 'Tools? Or Lawn and Garden?" Given that Sears is so heavily tool-oriented, you head in the direction of Tools.
When you reach the Tools department, you start looking at the signs at the end of each aisle.

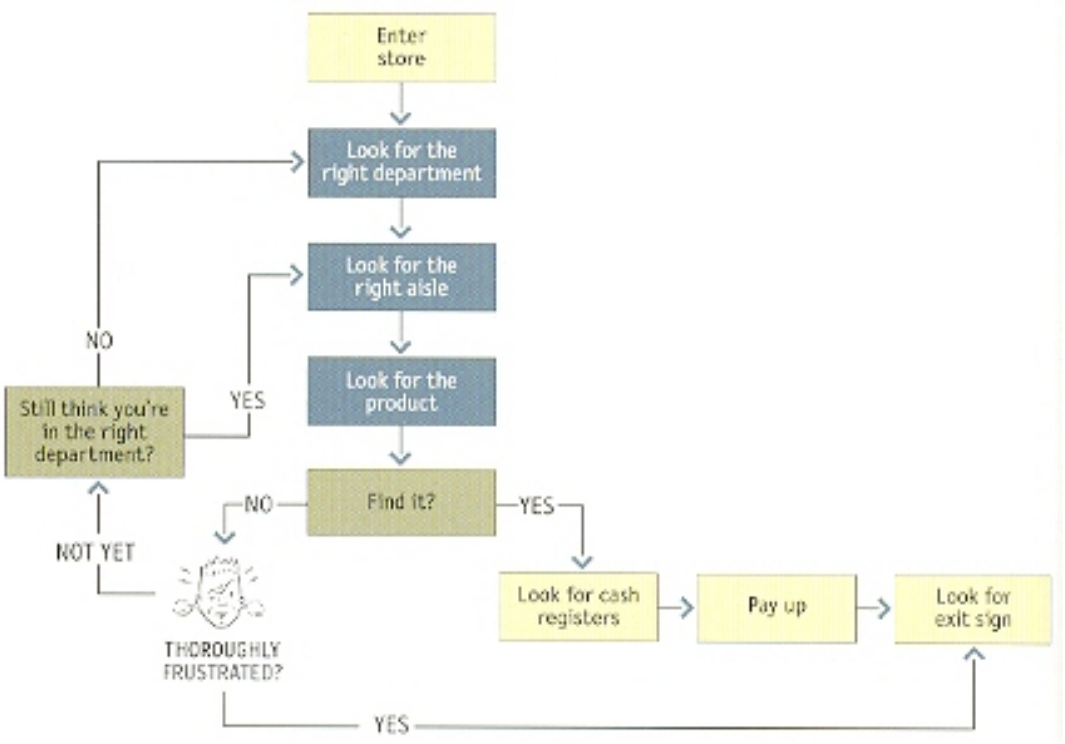


When you think you've got the right aisle, you start looking at the individual products.



© Copyright Virtual University of Pakistan

If it rums out you've guessed wrong, you try another aisle, or you may back up and start over again in the Lawn and Garden department. By the time you're done, the process looks something like this:



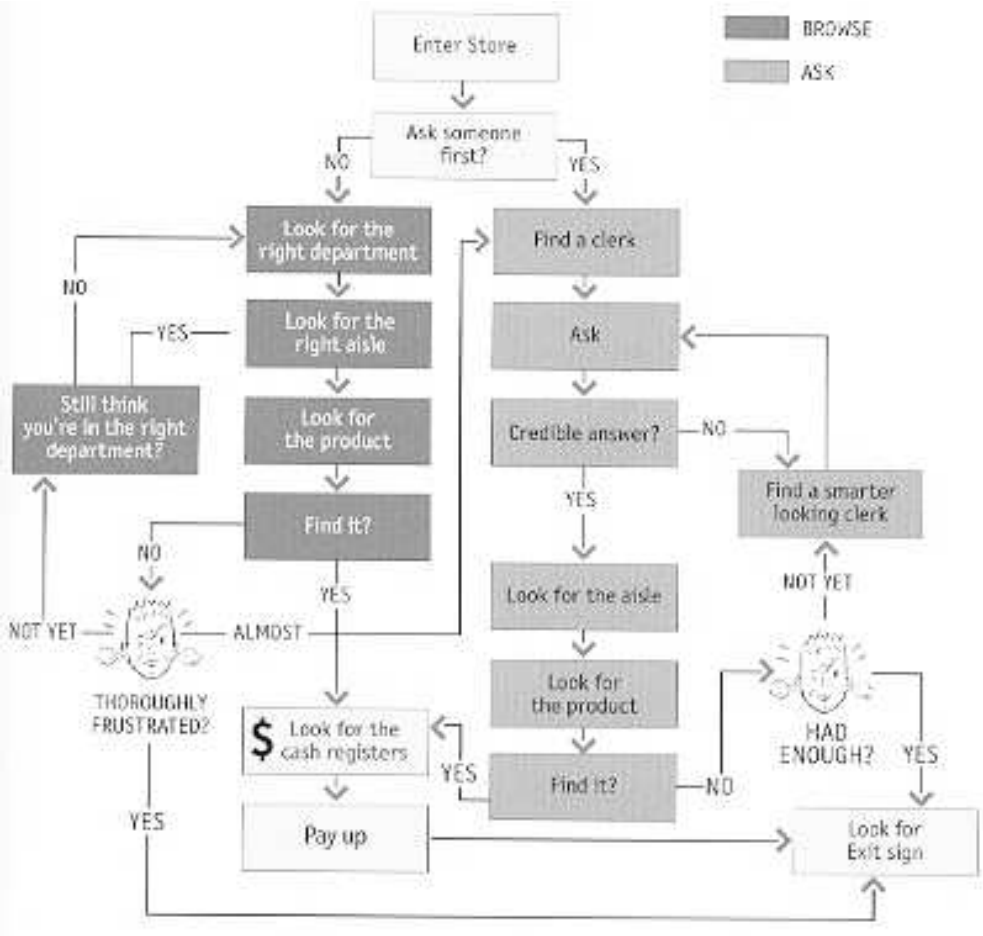
Basically, you use the store's navigation systems (the signs and the organizing hierarchy that the signs embody) and your ability to scan shelves full of products to find what you're looking for.

Of course, the actual process is a little more complex. For one thing, as you walk in the door you usually devote a few microseconds to a crucial decision: Are you going to start by looking for chainsaws on your own or are you going to ask someone where they are?
It's a decision based on a number of variables—how familiar you are with the store, how much you trust their ability to organize things sensibly, how much of a hurry you're in, and even how sociable you are.

When we factor this decision in, the process looks something like shown in figure on next page:

Notice that even if you start looking on your own, if things don't pan out there's a good chance that eventually you'll end up asking someone for directions anyway.

© Copyright Virtual University of Pakistan



32.2 Web Navigation

In many ways, you go through the same process when you enter a Web site.

- **You're usually trying to find something.**

  In the "real" world it might be the emergency room or a can of baked beans. On the Web, it might be the cheapest 4-head VCR with Commercial Advance or the name of the actor in Casablanca who played the headwaiter at Rick's.

- **You decide whether to ask first or browse first.**

  The difference is that on a Web site there's no one standing around who can tell you where things are. The Web equivalent of asking directions is searching—typing a description of what you're looking for in a search box and getting back a list of links to places where it *might* be.

Some people (Jakob Nielsen calls them "search-dominant" users) will almost always look for a search box as soon as they enter a site. (These may be the same people who look for the nearest clerk as soon as they enter a store.)
Other people (Nielsen's "link-dominant" users) will almost always browse first, searching only when they've run out of likely links to click or when they have gotten sufficiently frustrated by the site.

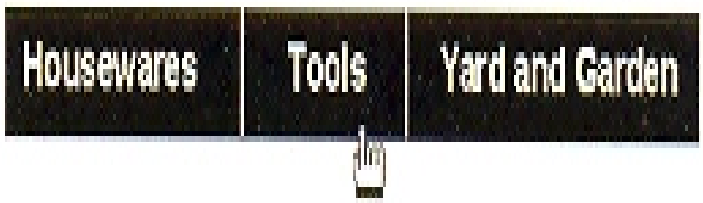
© Copyright Virtual University of Pakistan

For everyone else, the decision whether to start by browsing or searching depends on their current frame of mind, how much of a hurry they're in, and whether the site appears to have decent, browsable navigation.

- **If you choose to browse, you make your way through a hierarchy, using signs to guide you.**

  Typically you'll look around on the Home page for a list of the site's main sections (like the store's department signs) and elide on the one that seems right.

Then you will choose from the list of subsections.



With any luck, after another click or two you'll end up with a list of the kind of thing you're looking for:



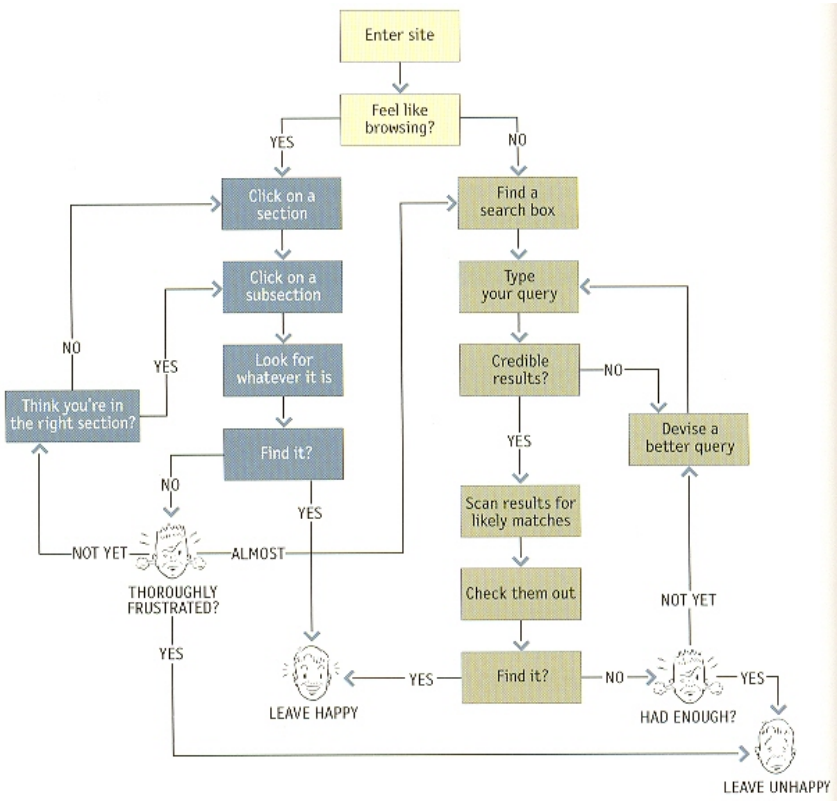
Then you can click on the individual links to examine them in detail, the same way you'd take products off the shelf and read the labels.

- **Eventually, if you can't find what you're looking for, you'll leave**.

  This is as true on a Web site as it is at Sears. You'll leave when you're convinced they haven't got it, or when you're just too frustrated to keep looking.

Here is what the process looks like:

© Copyright Virtual University of Pakistan



The unbearable lightness of browsing

Looking for things on a Web site and looking for them in the "real" world have a lot of similarities. When we're exploring the Web, in some ways it even *feels* like we're moving around in a physical space. Think of the words we use to describe the experience—like "cruising," "browsing," and "surfing." And clicking a link doesn't "load" or "display" another page—it "takes you to" a page.

But the Web experience is missing many of the cues we've relied on all our lives to negotiate spaces. Consider these oddities of Web space:

No sense of scale.

Even after we've used a Web site extensively, unless it's a very small site we tend to have very little sense of how big it is (50 pages? 1,000? 17,000?). For all we know, there could be huge corners we've never explored. Compare this to a magazine, a museum, or a department store, where you always have at least a rough sense of the seen/unseen ratio.

The practical result is that it's very hard to know whether you've seen everything of interest in a site, which means it's hard to know when to stop looking.

No sense of direction.

In a Web site, there's no left and right, no up and down. We may talk about moving up and down, but we mean up and down hi the hierarchy—to a more general or more specific level.

© Copyright Virtual University of Pakistan

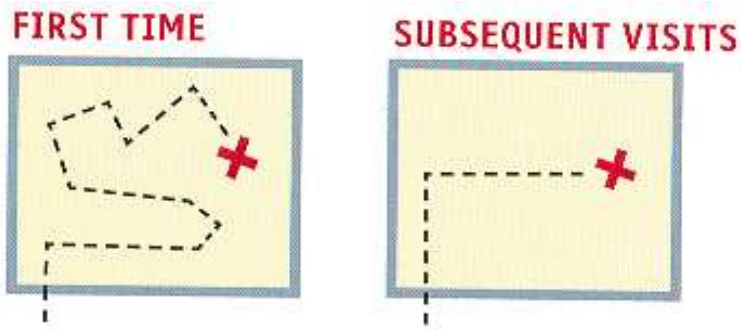
No sense of location.

In physical spaces, as we move around we accumulate knowledge about the space. We develop a sense of where things are and can take shortcuts to get to them.

We may get to the chainsaws the first time by following the signs, but the next time we're just as likely to think,

> "Chainsaws? Oh, yeah, I remember where they were: right rear corner, near the refrigerators."

And then head straight to them.



But on the Web, your feet never touch the ground; instead, you make your way around by clicking on links. Click on "Power Tools" and you're suddenly teleported to the Power Tools aisle with no traversal of space, no glancing at things along the way.

When we want to return to something on a Web site, instead of relying on a physical sense of where it is we have to remember where it is in the conceptual hierarchy and retrace our steps.

This is one reason why bookmarks—stored personal shortcuts—are so important, and why the Back button accounts for somewhere between 30 and 40 percent of all Web clicks.

It also explains why the concept of Home pages is so important. Home pages are—comparatively—fixed places. When you're in a site, the Home page is like the North Star. Being able to click Home gives you a fresh start.

This lack of physicality is both good and bad. On the plus side, the sense of weightlessness can be exhilarating, and partly explains why it's so easy to lose track of time on the Web—the same as when we're "lost" in a good book.

On the negative side, I think it explains why we use the term "Web navigation" even though we never talk about "department store navigation" or "library navigation." If you look up *navigation* in a dictionary, it's about doing two things: getting from one place to another, and figuring out where you are.

We talk about Web navigation because "figuring out where you are" is a much more pervasive problem on the Web than in physical spaces. We're inherently-lost when we're on the Web, arid we can't peek over the aisles to see where we are. Web navigation compensates for this missing sense of place by embodying the site's hierarchy, creating a sense of "there."

Navigation isn't just a *feature* of a Web site; it is the Web site, in the same way that the building, the shelves, and die cash registers *are* Sears. Without it, there's no *there* there.

The moral? Web navigation had better be good.

© Copyright Virtual University of Pakistan

The overlooked purposes of navigation

Two of the purposes of navigation are fairly obvious: to help us find whatever it is we're looking for, and to tell us where we are.
And we've just talked about a third:

It gives us something to hold on to.

As a rule, it's no fun feeling lost. (Would you rather "feel lost" or "know your way around?"} Done right, navigation puts ground under our feet (even if it's virtual ground) and gives us handrails to hold on to— to make us feel grounded.
But navigation has some other equally important—and easily overlooked—functions:

It tells us what's here.

By making the hierarchy visible, navigation tells us what the site contains. Navigation reveals content! And revealing the site may be even more important than guiding or situating us.

It tells us how to use the site.

If the navigation is doing its job, it tells you *implicitly* where to begin and what your options are. Done correctly, it should be all the instructions you need. (Which is good, since most users will ignore any other instructions anyway.)

It gives us confidence in the people who built it.

Every moment we're in a Web site, we're keeping a mental running tally: "Do these guys know what they're doing?" It's one of the main factors we use in deciding whether to bail out and deciding whether to ever come back. Clear, well-thought-out navigation is one of the best opportunities a site has to create a good impression.

Web navigation conventions

Physical spaces like cities and buildings (and even information spaces like books and magazines) have their own navigation systems, with conventions that have evolved over time like street signs, page numbers, and chapter titles. The conventions specify (loosely) the appearance and location of the navigation elements so we know what to look for and where to look when we need them.
Putting them in a standard place lets us locate them quickly, with a minimum of effort; standardizing their appearance makes it easy to distinguish them from everything else.
For instance, we expect to find street signs at street corners, we expect to find them by looking up (not down), and we expect them to look like street signs (horizontal, not vertical).



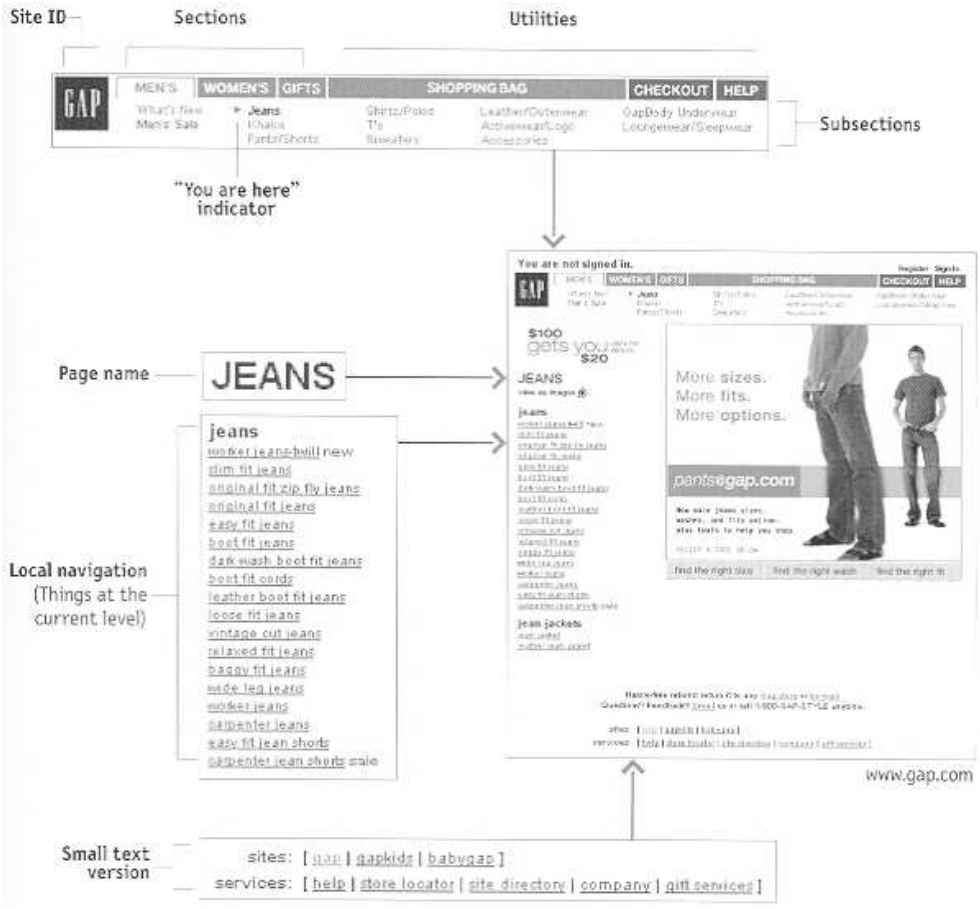
We also take it for granted that the name of a building will be above or next to its front door. In a grocery store, we expect to find signs near the ends of each aisle. In a magazine, we

© Copyright Virtual University of Pakistan

know there will be a table of contents somewhere in the first few pages and page numbers somewhere in the margin of each page—and that they'll look like a table of contents and page numbers.

Think of how frustrating it is when one of these conventions is broken (when magazines don't put page numbers on advertising pages, for instance).

Navigation conventions for the Web have emerged quickly, mostly adapted from existing print conventions. They'll continue to evolve, but for the moment these are the basic elements:



© Copyright Virtual University of Pakistan

Don't look now, but it's following us

Web designers use the term *penitent navigation* (or *global navigation)* to describe the set of navigation elements that appear on every page of a site.
Done right, persistent navigation should say—preferably in a calm, comforting voice:
"The navigation is over here. Some parts will change a little depending on where you are, but it will always be here, and it will always work the same way."
Just having the navigation appear in the same place on every page with a consistent look gives you instant confirmation that you're still in the same site—which is more important than you might think. And keeping it the same throughout the site means that (hopefully) you only have to figure out how it works once.
Persistent navigation should include the five elements you most need to have on hand at all times.
We'll look at each of them in a minute.

Some Exceptions

There are two exceptions to the "follow me everywhere" rule:

The Home page.

The Home page is not like the other pages—it has different burdens to bear, different promises to keep. As we'll see in the next chapter, this sometimes means that it makes sense not to use the persistent navigation there.

Forms.

On pages where a form needs to be filled in, the persistent navigation can sometimes be an unnecessary distraction. For instance, when I'm paying for my purchases on an e-commerce site you don't really want me to do anything but finish filling in the forms. The same is true when I'm registering, giving feedback, or checking off personalization preferences.
For these pages, it's useful to have a minimal version of the persistent navigation with just the Site ID, a link to Home, and any Utilities that might help me fill out the form.

Site ID

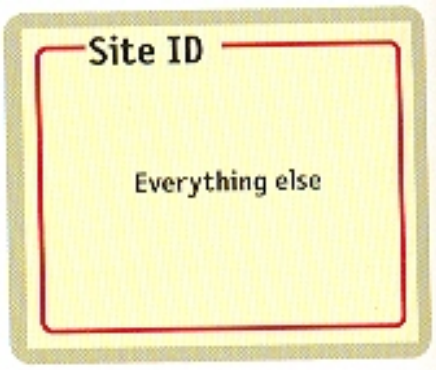
The Site ID or logo is like the building name for a Web site. At Sears, I really only need to see the name on my way in; once I'm inside, I *know* I'm still in Sears until I leave. But on the Web—where my primary mode of travel is teleportation—I need to see it on every page.

In the same way that we expect to see the name of a building over the front entrance, we expect to see the Site ID at the top of the page—usually in (or at least near] the upper left corner/
Why? Because the Site ID represents the whole site, which means it's the highest thing in the logical hierarchy of the site.

© Copyright Virtual University of Pakistan

This site
Sections of this site
Subsections
Sub-subsections, etc.
This page
Areas of this page
Items on this page

And there are two ways to get this primacy across in the visual hierarchy of the page: either make it the most prominent thing on the page, or make it frame everything else.
Since you don't want the ID to be the most prominent element on the page (except, perhaps, on the Home page), the best place for it—the place that is least likely to make me think—is at the top, where it frames the entire page.



And in addition to being where we would expect it to be, the Site ID also needs to *look* like a Site ID. This means it should have the attributes we would expect to see in a
brand logo or the sign outside a store: a distinctive typeface, and a graphic that's recognizable at any size from a button to a billboard.



The Sections

The Sections—sometimes called the *primary navigation*—are the links to the main sections of the site: the fop level of the site's hierarchy



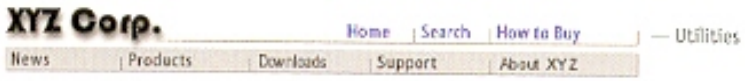
In most cases, the persistent navigation will also include space to display the *secondary* navigation: the list of subsections in the current section.



© Copyright Virtual University of Pakistan

The Utilities

Utilities are the links to important elements of the site that aren't reajiy part of the content hierarchy.



These are things that either can help me use the site (like Help, a Site Map, or a Shopping Cart} or can provide information about its publisher (like About Us arid Contact Us).

Like the signs for the facilities in a store, the Utilities list should be slightly less prominent than the Sections.



Utilities will vary for different types of sites. For a corporate or e-commerce site, for example, they might include any of the following:

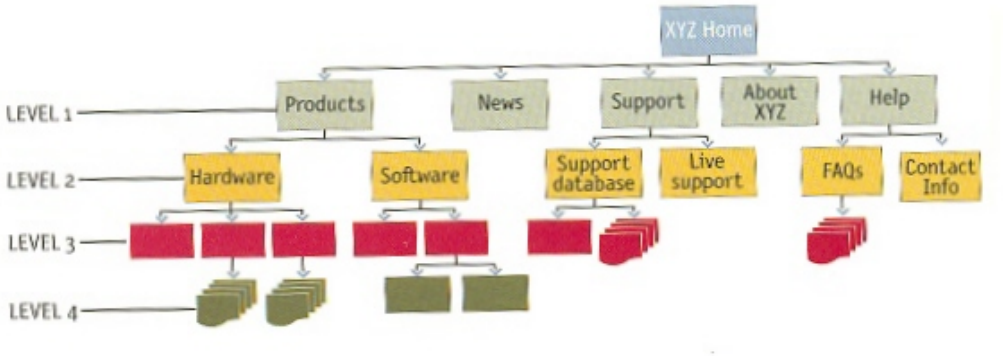
About Us	Downloads	How to Shop	Register
Archives	Directory	Jobs	Search
Checkout	Forums	My _____	Shopping Cart
Company Info	FAQs	News	Sign in
Contact Us	Help	Order Tracking	Site Map
Customer Service	Home	Press Releases	Store Locator
Discussion Boards	Investor Relations	Privacy Policy	Your Account

As a rule, the persistent navigation can accommodate only four or five Utilities—the tend to get lost in the crowd. The less frequently used leftovers can be grouped together on the Home page.

Low Level Navigation

It's happened so often I've come to expect it: When designers I haven't worked with before send me preliminary page designs so I can check for usability issues. I almost inevitably get a flowchart that shows a site four levels deep...
...and sample pages for the Home page and the top *two* levels.

© Copyright Virtual University of Pakistan



I keep flipping the pages looking for more, or at least for the place where they've scrawled, "Some magic happens here," but I never find even that. I think this is one of the most common problems in Web design (especially in larger sites): failing to give the lower-level navigation the same attention as the top. In so many sites, as soon as you get past the second level, the navigation breaks down and becomes *ad hoc*. The problem is so common that it's actually hard to find good examples of third-level navigation.
Why does this happen?

Partly, because good multi-level navigation is just plain hard to figure out given the limited amount of space on the page, and the number of elements that have to be squeezed in. Partly because designers usually don't even have enough time to figure out the first two levels.
Partly because it just doesn't seem that important. (After all, how important can it be? It's not primary. It's not even secondary.) And there's a tendency to think that by the time people get that far into the site, they'll understand how it works.
And then there's the problem of getting sample content and hierarchy examples for lower-level pages. Even if designers ask, they probably won't get them, because the people responsible for the content usually haven't thought things through that far, either.
But the reality is that users usually end up spending as much time on lower-level pages as they do at the top. And unless you've worked out top-to-bottom navigation from the beginning, it's very hard to graft it on later and come up with something consistent.
The moral? It's vital to have sample pages that show the navigation for all the potential levels of the site before you start arguing about the color scheme for the Home page.

Page names

If you've ever spent time in Los Angeles, you understand that it's not just a song lyric—L.A. really is a great big freeway. And because people in LA. take driving seriously, they have the best street signs I've ever seen. In L.A.,

- Street signs are big. When you're stopped at an intersection, you can read the sign for the next cross street.
- They're in the right place—hanging *ovsr* the street you're driving on, so all you have to do is glance up.

© Copyright Virtual University of Pakistan

Now, I'll admit I'm a sucker for this kind of treatment because I come from Boston, where you consider yourself lucky if you can manage to read the street sign while there's still time to make the turn.



The result? When I'm driving in LA., *I* devote less energy and attention to dealing with where I am and more to traffic, conversation, and listening to *All Things Considered.*

Page names are the street signs of the Web. Just as with street signs, when things are going well I may not notice page names at all. But as soon as I start to sense that I may not be headed in the right direction, I need to be able to spot the page name effortlessly so I can get my bearings.

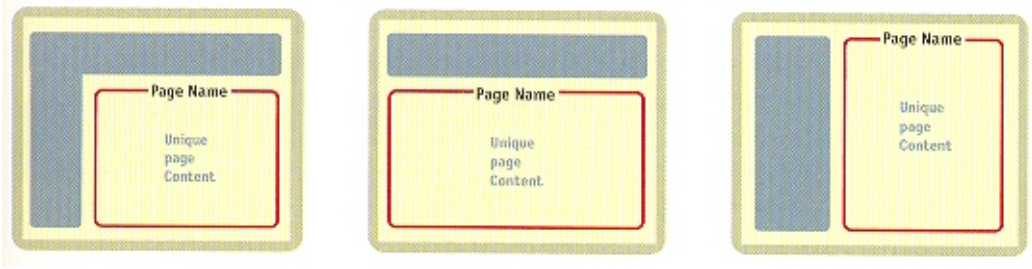
There are four things you need to know about page names:

- Every page needs a name. Just as every corner should have a street sign, every page should have a name.



Designers sometimes think, "Well, we've highlighted the page name in the navigation. That's good enough." It's a tempting idea because it can save space, and it's one less element to work into the page layout, but it's not enough. You need a page name, too.

- **The name needs to be in the right place.** In the visual hierarchy of the page, the page name should appear to be framing the content that is unique to this page. (After all, that's what it's naming—not the navigation or the ads, which are just the infrastructure.)

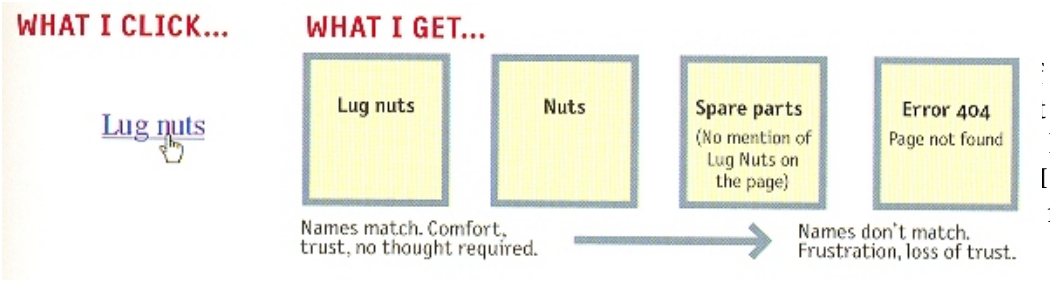


- **The name needs to be prominent**. You want the combination of position, size, color, and typeface to make the name say "This is the heading for the entire page." In most cases, it will be the largest text on the page.
- **The name needs to match what I clicked**. Even though nobody ever mentions it, every site makes an implicit social contract with its visitors:

© Copyright Virtual University of Pakistan

In other words, if" I click on a link or button that says "Hot mashed potatoes," the site will take me to a page named "Hot mashed potatoes."

It may seem trivial, but it's actually a crucial agreement. Each time a site violates it, I'm forced to think, even if only for milliseconds, "Why are those two things different?" And if there's a major discrepancy between the link name and the page name or a lot of minor discrepancies, my trust in the site—and the competence of the people who publish it—will be diminished.



the
(a)
For
I get
feel

"You are here"

One of the ways navigation can counteract the Web's inherent "lost in space" feeling is by showing me where I am in the scheme of things, the same way that a "You are here" indicator does on the map in a shopping mall—or a National Park.



On the Web, this is accomplished by highlighting my current location in whatever navigational bars, lists, or menus appear on the page.



© Copyright Virtual University of Pakistan

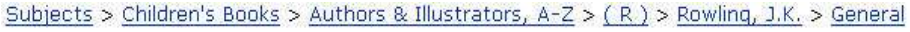
In this example, the current section (Women's) and subsection (Pants/Shorts) have both been "marked." There are a number of ways to make the current location stand out:



ey need to stand out; if they don't, they lose their value as visual cues and end up just adding more noise to the page. One way to ensure that they stand out is to apply more than one visual distinction—for instance, a different color *and* bold text.

Breadcrumbs

Like "You are here" indicators, Breadcrumbs show you where you are. (Sometimes they even include the words "You are here.")



They're called Breadcrumbs because they're reminiscent of the trail of crumbs Hansel dropped in the woods so he and Gretel could End their way back home.

Unlike "You are here" indicators, which show you where you are in the context of the site's hierarchy, Breadcrumbs only show you the path from the Home page to where you are. (One shows you where you are in the overall scheme of things, the other shows you how to get there—kind of like the difference between looking at a road map and looking at a set of turn-by-turn directions. The directions can be very useful, but you can learn more from the map.)

You could argue that bookmarks are more like the fairy tale breadcrumbs, since we drop them as we wander, in anticipation of possibly wanting to retrace our steps someday. Or you could say that visited links (links that have changed color to show that you've clicked on them) are more like breadcrumbs since they mark the paths we've taken, and if we don't revisit them soon enough, our browser (like the birds) will swallow them up.

© Copyright Virtual University of Pakistan

Lecture 33.

Evaluation V

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the use of tabs
- Conduct the trunk test

Four reasons to use tabs

It hasn't been proven (yet), but It is strongly suspected that Leonardo da Vinci invented tab dividers sometime in the late I5th century. As interface devices go, they're clearly a product of genius.



Tabs are one of the very few cases where using a physical metaphor in a user interface actually works. Like the tab dividers in a three-ring binder or tabs on folders in a file drawer, they divide whatever they're sticking out of into sections. And they make it easy to open a section by reaching for its tab (or, in the case of the Web, clicking on it).
In the past year, the idea has really caught on, and many sites have started using tabs for navigation. They're an excellent navigation choice for large sites. Here's why:

- **They're self-evident.** I've never seen anyone—no matter how "computer illiterate"—look at a tabbed interface and say, "Hmmm. I wonder what *those* do?"
- **They're hard to miss.** When I do point-and-click user tests, I'm surprised at how often people can overlook button bars at the top of a Web page. But because tabs are so visually distinctive, they're hard to overlook. And because they're hard to mistake for anything *but* navigation, they create the kind of obvious-at-a-glance division you want between navigation and content.
- **They're slick.** Web designers are always struggling to make pages more visually interesting without making them slow to load. If done correctly (see below), tabs can add polish *and* serve a useful purpose, all without bulking up the page size.
- **They suggest a physical space**. Tabs create the illusion that the active tab physically moves to the front.

It's a cheap trick, but effective, probably because it's based on a visual cue that we're very good at detecting ("things in front of other things"). Somehow, the result is a stronger-than-usual sense that the site is divided into sections and that you're *in* one of the sections.

Why Amazon is good?

As with many other good Web practices, Amazon was one of the first sites to use tab dividers for navigation, and the first to really get them right. Over time, they've continued

© Copyright Virtual University of Pakistan

to tweak and polish their implementation to the point where it's nearly perfect, even though they keep adding tabs as they expand into different markets.



Anyone thinking of using tabs should look carefully at what Amazon has done over the years, and slavishly imitate these tour key attributes of their tabs:

- **They'**re **drawn correctly.** For tabs to work to full effect, the graphics have to create the visual illusion that the active tab is *in front* o/the other tabs. This is the main thing that makes them feel like tabs—even more than the distinctive tab shape. '

To create this illusion, the active tab needs to be a different color or contrasting shade, and it has to physically connect with the space below it. This is what makes the active tab "pop" to the front.

- **They load fast**. Amazon's single row of tabs required only two graphics per page, totaling less than 6k—including the logo!

Some sites create tabs (or any kind of button bar, for that matter) by using a separate graphic for each button—piecing them together like a patchwork quilt. If Amazon did it this way, the 17 pieces would look like this:
This is usually done for two reasons:

- **Rollovers.** To implement rollovers in tabs or button bars, each button needs to be a separate graphic. Rollovers have merit, but in most cases I don't think they pull their weight.
- A misguided belief that it will be faster. The theory is that after the first page is loaded, most of the pieces will be cached on the user's hard drive,'[1,0] so as the user moves from page to page the browser will only have to download the small pieces that change and not the entire site.

It's an attractive theory, but the reality is that it usually means that users end up with a longer load time on the first page they see. Which is exactly where you *don't* want it.
And even if the graphics are cached, the browser still has to send a query- to the server for each graphic to make sure it hasn't been updated. If the server is at all busy, the result is a visually disturbing crazy-quilt effect as the pieces load on ever}' page.

- **They're color coded**. Amazon uses a different tab color for each section of the site, and they use the same color in the other navigational elements on the page to tie them all together.

Color coding of sections is a very good idea—as long as you don't count on everyone noticing it. Some people (roughly i out of 200 women and i out of 12 men—particularly over the age of 40) simply can't detect some color distinctions because of color-blindness.
More importantly, from what I've observed, a much larger percentage (perhaps as many as half) just aren't very *aware* of color coding in any useful way.
Color is great as an additional cue, but you should never rely on it as the *only* cue.

© Copyright Virtual University of Pakistan

Amazon makes a point of using fairly vivid, saturated colors that are hard to miss. And since the inactive tabs are neutral beige, there's a lot of contrast—which even color-blind users can detect—between them and the active tab.

- There's a tab selected when you enter the site. If there's no tab selected when I enter a site (as on Quicken.com, for instance), I lose the impact of the tabs in the crucial first few seconds, when it counts the most.

Amazon has always had a tab selected on their Home page. For a long time, it was the Books tab.
Eventually, though, as the site became increasingly less book-centric, they gave the Home page a tab of its own (labeled "Welcome").
Amazon had to create the Welcome tab so they could promote products from their other sections—not just books—on the Home page. But they did it at the risk of alienating existing customers who still think of Amazon as primarily a bookstore and hate having to click twice to get to the Books section. As usual, the interface problem is just a reflection of a deeper—and harder to solve—dilemma.
Here's how you perform the trunk test:
Step 1 Choose a page anywhere in the site at random, and print it.
Step 2 Hold it at arm's length or squint so you can't really study it closely.
Step 3 As quickly as possible, try to find and circle each item in the list below. (You won't find ail ot the items on every page.)

33.1 Try the trunk test

Now that you have a feeling for all of the moving parts, you're ready to try my acid test for good Web navigation. Here's how it goes:
Imagine that you've been blindfolded and locked in the trunk of a car, then driven around for a while and dumped on a page somewhere deep in the bowels of a Web site. If the page is well designed, when your vision clears you should be able to answer these questions without hesitation:

- What site is this? (Site ID)
- What page am I on? (Page name)
- What are the major sections of this site? (Sections)
- What are my options at this level? (Local navigation)
- Where am I in the scheme of things? ("You are here" indicators)
- How can I search?

Why the *Goodfellas* motif? Because it's so easy to forget that the Web experience is often more like being shanghaied than following a garden path. When you're designing pages, it's tempting to think that people will reach them by starting at the Home page and following the nice, neat paths you've laid out. But the reality is that we're often dropped down in the middle of a site with no idea where we are because we've followed a link from a search engine or from another site, and we've never seen this site's navigation scheme before.
And the blindfold? You want your vision to be slightly blurry, because the true test isn't whether you can figure it out given enough time and close scrutiny. The standard needs to be that these elements pop off the page so clearly that it doesn't matter whether you're

© Copyright Virtual University of Pakistan

looking closely or not. You want to be relying solely on the overall appearance of things, not the details.

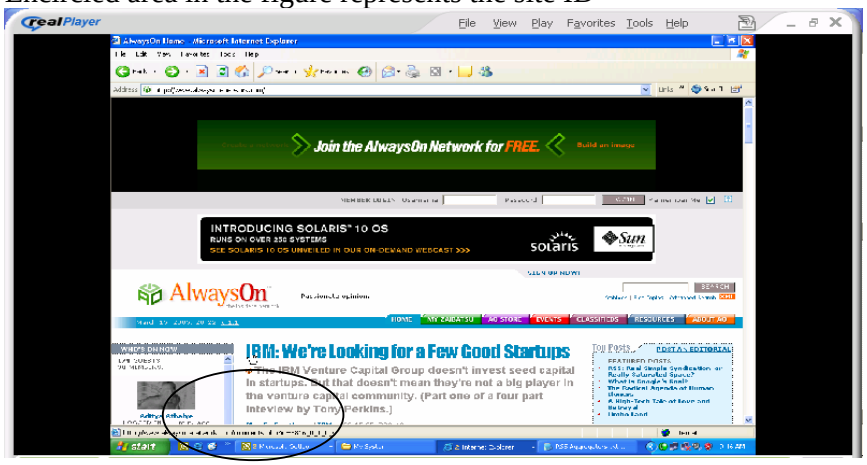
Here's one to show you how it's done.

CIRCLE:

1. Site 1D
2. Page name
3. Sections and subsections
4. Local navigation
5. "You are here" indicator(s)
6. Search

Site ID

Encircled area in the figure represents the site ID



Page Name

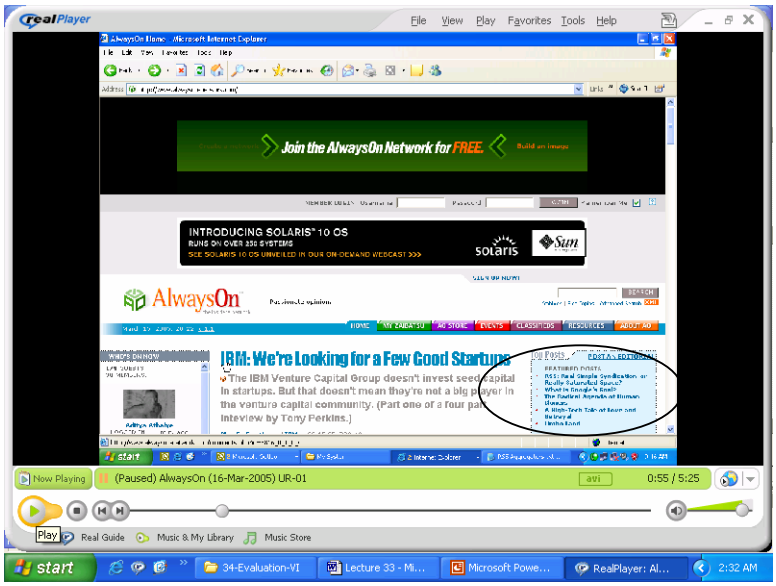
Encircled area in the figure represents the Page Name



Sections and subsections

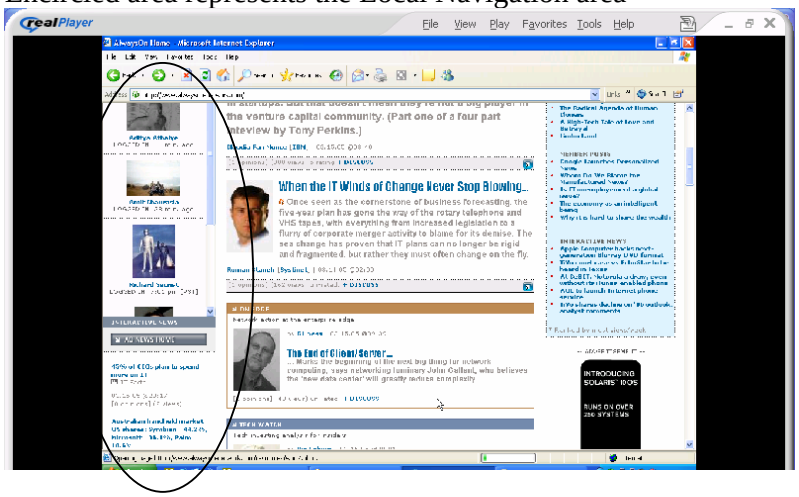
Encircled area in the figure represents the sections of this site

© Copyright Virtual University of Pakistan



Local Navigation

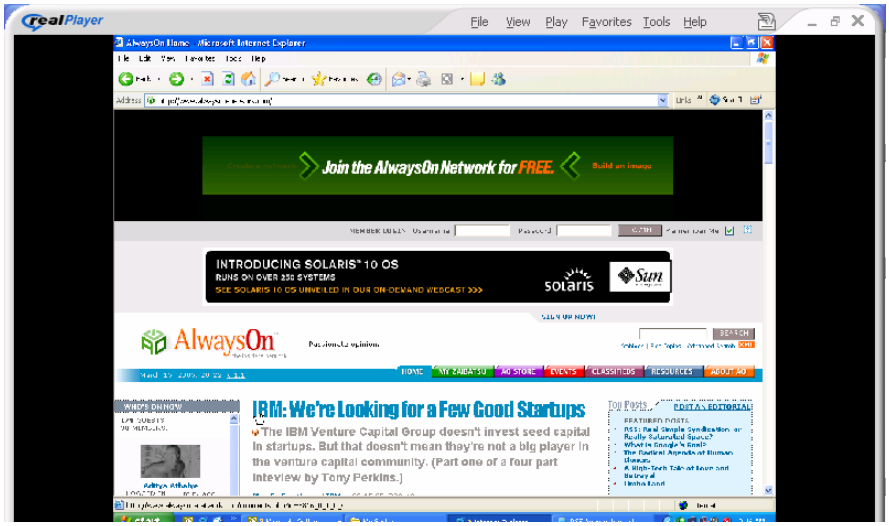
Encircled area represents the Local Navigation area



You are here indicator

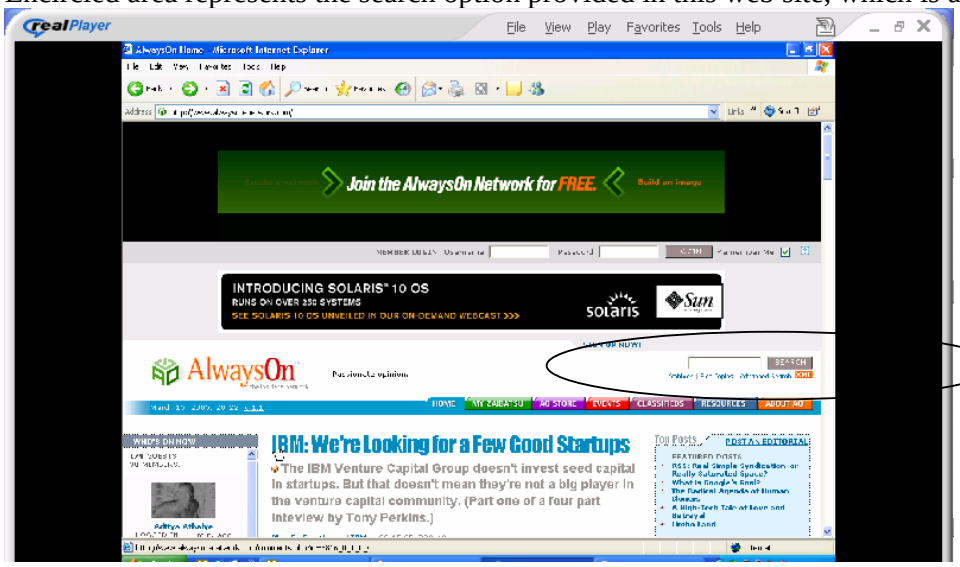
In this site as you can see the tabs are used to separate different sections. At this time we are on home section. This is indicated by the use of same color for back ground as the color of the tab.

© Copyright Virtual University of Pakistan



Search

Encircled area represents the search option provided in this web site, which is a Utility.



© Copyright Virtual University of Pakistan

Lecture 34.

Evaluation – Part VI

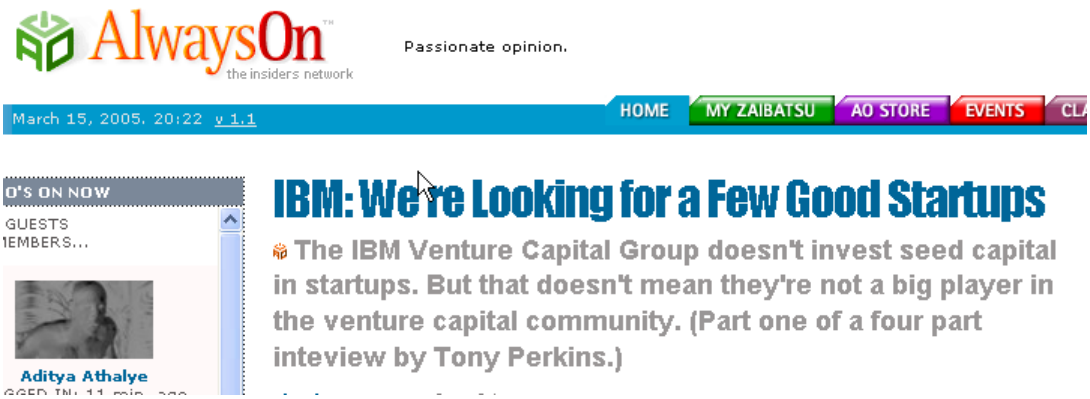
Learning Goals

The purpose of this lecture is to learn how to perform heuristic evaluations. This lecture involves looking at a web site to identify any usability issues.

Clear and readable fonts not being used

What font?

Color is dim

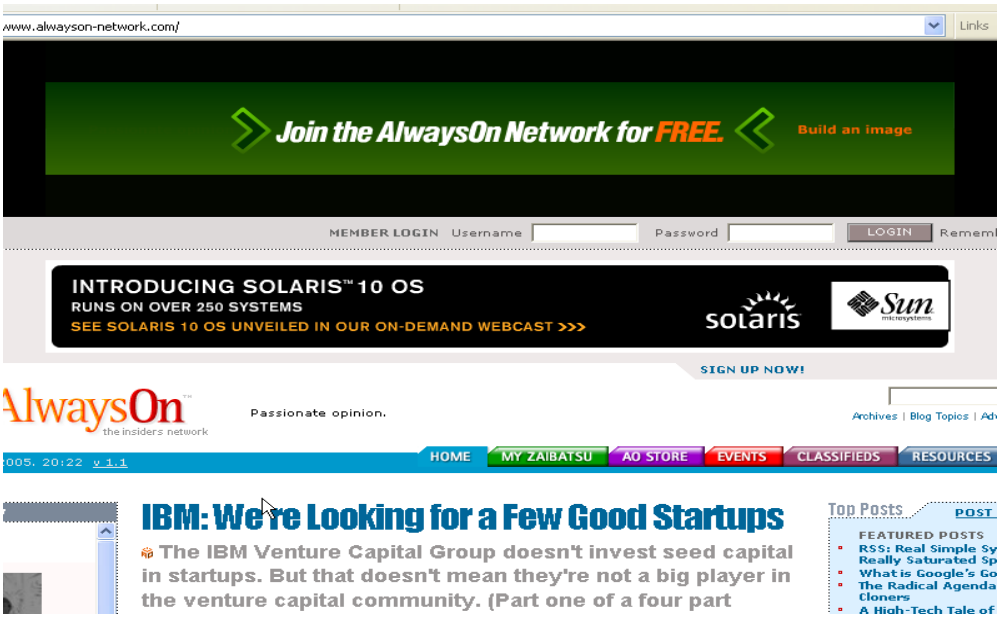


Browser Title always contains the word 'Home'

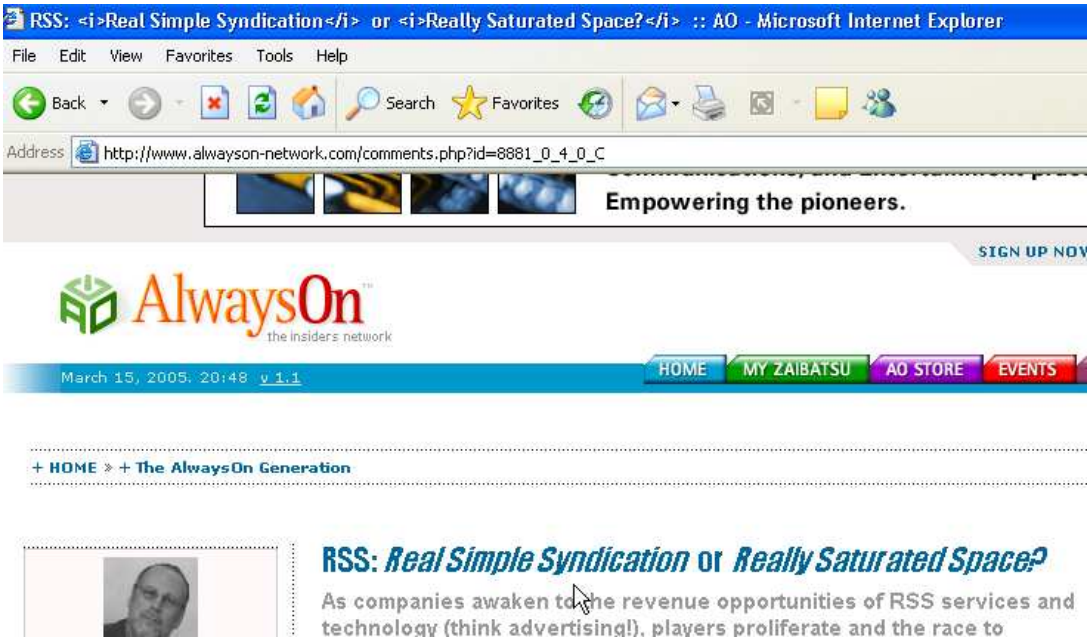


© Copyright Virtual University of Pakistan

Banner ads take up too much space



Invalid browser title characters:



Use of highlighted tabs in global navigation bar shows this is the 'Home' page

© Copyright Virtual University of Pakistan



Absence of highlighted tab confuses user about the current section being viewed



Version numbers should not be given on the main page of web site since it does not interest users



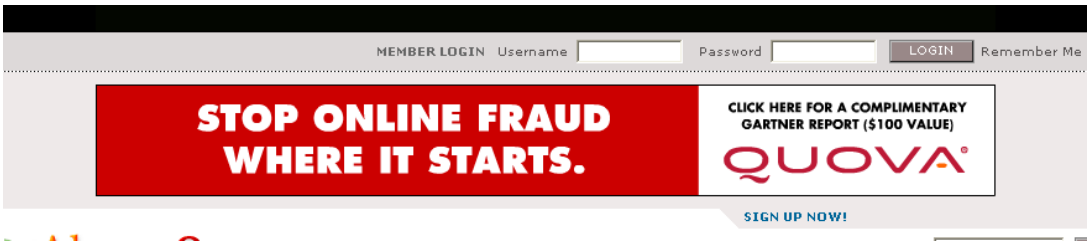
Breadcrumbs format do not follow standard conventions





‘Sign up now’ links appears to refer to free report …

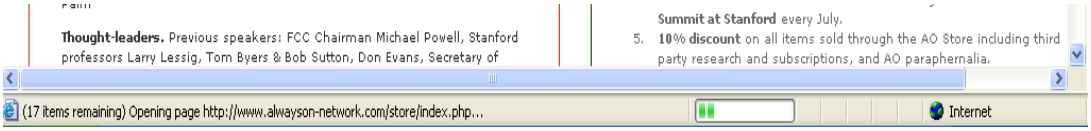
© Copyright Virtual University of Pakistan



… but the 'sign up now' link actually takes the user to the online store



The page has horizontal scrolling



© Copyright Virtual University of Pakistan

Lecture 35.

Evaluation – Part VII

Learning Goals

- The aim of this lecture is to understand the strategic nature of usability
- The aim of this lecture is to understand the nature of the Web

35.1 The relationship between evaluation and usability?

With the help of evaluation we can uncover problems in the interface that will help to improve the usability of the product.

Questions to ask

- Do you understand the users?
- Do you understand the medium?
- Do you understand the technologies?
- Do you have commitment?

Technologies

- You must understand the constraints of technology
- What can we implement using current technologies
- Building a good system requires a good understanding of technology constraints and potentials

Users

- Do you know your users?
- What are their goals and behaviors?
- How can they be satisfied?
- Use goals and personas

Commitment

- Building usable systems requires commitment?
- Do you have commitment at every level in your organization?

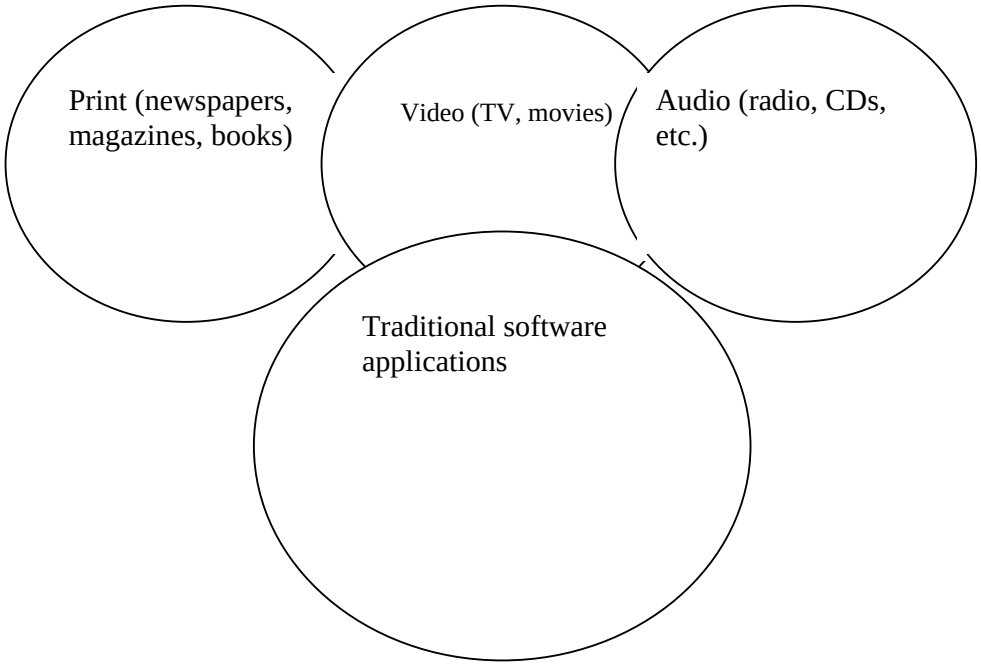
Medium

- You must understand the medium that you are working in to build a good usable system

© Copyright Virtual University of Pakistan

Nature of the Web Medium

The World Wide Web is a combination of many different mediums of communication.



It would be true to say that the Web is in fact a super medium which incorporates all of the above media.

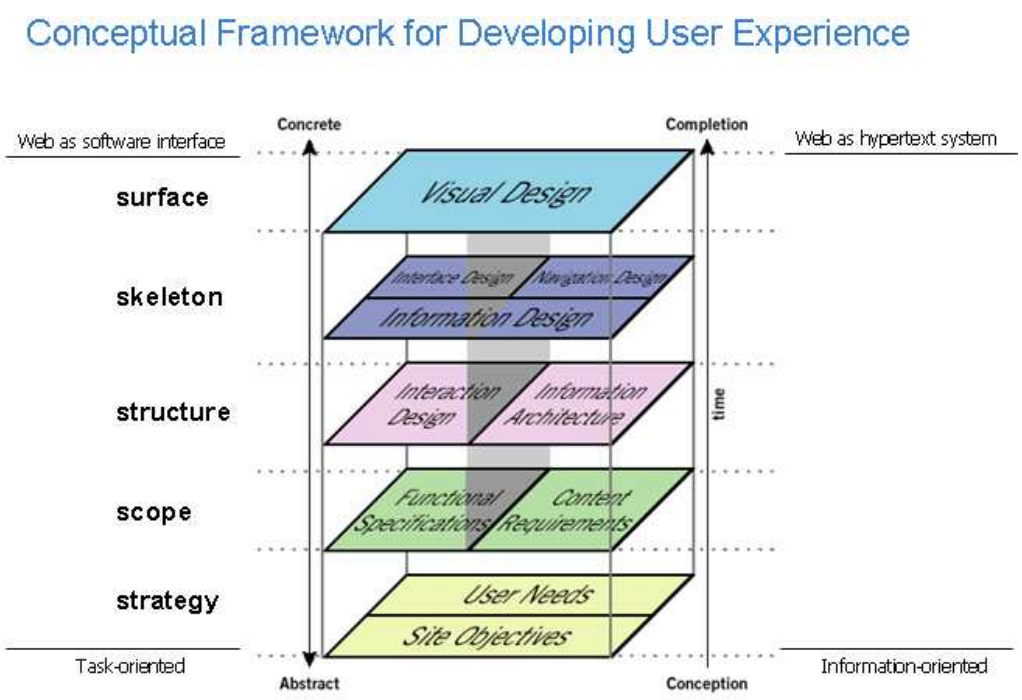
Today's we pages and applications incorporate elements of the following media:

- Print
- Video
- Audio
- Software applications

Because of its very diverse nature, the Web is a unique medium and presents many challenges for its designers.

We can more clearly understand the nature of the Web by looking at a conceptual framework.

© Copyright Virtual University of Pakistan



The Surface Plane

On the surface you see a series of Web pages, made up of images and text. Some of these images are things you can click on, performing some sort of function such as taking you to a shopping cart. Some of these images are just illustrations, such as a photograph of a book cover or the logo of the site itself.

The Skeleton Plane

Beneath that surface is the skeleton of the site: the placement of buttons, tabs, photos, and blocks of text. The skeleton is designed to optimize the arrangement of these elements for maximum effect and efficiency-so that you remember the logo and can find that shopping cart button when you need it.

The Structure Plane

The skeleton is a concrete expression of the more abstract structure of the site. The skeleton might define the placement of the interface elements on our checkout page; the structure would define how users got to that page and where they could go when they were finished there. The skeleton might define the arrangement of navigational items allowing the users to browse categories of books; the structure would define what those categories actually were.

The Scope Plane

The structure defines the way in which the various features and functions of the site fit together. Just what those features and functions are constitutes the scope of the site. Some sites that sell books offer a feature that enables users to save previously used addresses so they can be used again. The question of whether that feature-or any feature-is included on a site is a question of scope.

© Copyright Virtual University of Pakistan

The Strategy Plane

The scope is fundamentally determined by the strategy of the site. This strategy incorporates not only what the people running the site want to get out of it but what the users want to get out of the site as well. In the case of our bookstore example, some of the strategic objectives are pretty obvious: Users want to buy books, and we want to sell them. Other objectives might not be so easy to articulate.

Building from Bottom to Top

These five planes-strategy, scope, structure, skeleton, and surface¬provide a conceptual framework for talking about user experience problems and the tools we use to solve them. To further complicate matters, people will use the same terms in dif¬ferent ways. One person might use "information design" to refer to what another knows as "information architecture." And what's the difference between "interface design" and "interaction design?" Is there one?

Fortunately, the field of user experience seems to be moving out of this Babel-like state. Consistency is gradually creeping into our dis¬cussions of these issues. To understand the terms themselves, how ever, we should look at where they came from.

When the Web started, it was just about hypertext. People could create documents, and they could link them to other documents. Tim Berners-Lee, the inventor of the Web, created it as a way for researchers in the high-energy physics community, who were spread out all over the world, to share and refer to each other's findings. He knew the Web had the potential to be much more than that, but few others really understood how great its potential was.

People originally seized on the Web as a new publishing medium, but as technology advanced and new features were added to Web browsers and Web servers alike, the Web took on new capabilities. After the Web began to catch on in the larger Internet community, it developed a more complex and robust feature set that would enable Web sites not only to distribute information but to collect and manipulate it as well. With this, the Web became more interactive, responding to the input of users in ways that were very much like traditional desktop applications.

With the advent of commercial interests on the Web, this application functionality found a wide range of uses, such as electronic commerce, community forums, and online banking, among others. Meanwhile, the Web continued to flourish as a publishing medium, with countless newspaper and magazine sites augmenting the wave of Web-only "e-zines" being published. Technology continued to advance on both fronts as all kinds of sites made the transition from static collections of information that changed infrequently to dynamic, database-driven sites that were constantly evolving.

When the Web user experience community started to form, its members spoke two different languages. One group saw every problem as an application design problem, and applied problem-solving approaches from the traditional desktop and mainframe software worlds. (These, in turn, were rooted in common practices applied to creating all kinds of products, from cars to running shoes.) The other group saw the Web in terms of information distribution and retrieval, and applied problem-solving approaches from the traditional worlds of publishing, media, and information science.

This became quite a stumbling block. Very little progress could be made when the community could not even agree on basic terminology. The waters were further muddied by the fact that many Web sites could not be neatly categorized as either applications or

© Copyright Virtual University of Pakistan

hypertext information spaces-a huge number seemed to be a sort of hybrid, incorporating qualities from each world.

To address this basic duality in the nature of the Web, let's split our five planes down the middle. On the left, we'll put those elements specific to using the Web as a software interface. On the right, we'll put the elements specific to hypertext information spaces.

On the software side, we are mainly concerned with tasks-the steps involved in a process and how people think about completing them. Here, we consider the site as a tool or set of tools that the user employs to accomplish one or more tasks. On the hypertext side, our concern is information-what information the site offers and what it means to our users. Hypertext is about creating an information space that users can move through.

The Elements of User Experience

Now we can map that whole confusing array of terms into the model. By breaking each plane down into its component elements, we'll be able to take a closer look at how all the pieces fit together to create the whole user experience.

The Strategy Plane

The same strategic concerns come into play for both software products and information spaces. User needs are the goals for the site that come from outside our organization-specifically from the people who will use our site. We must understand what our audience wants from us and how that fits in with other goals it has.

Balanced against user needs are our own objectives for the site. These site objectives can be business goals ("Make $1 million in sales over the Web this year") or other kinds of goals ("Inform voters about the candidates in the next election").

The Scope Plane

On the software side, the strategy is translated into scope through the creation of functional specifications: a detailed description of the "feature set" of the product. On the information space side, scope takes the form of content requirements: a description of the various content elements that will be required.

The Structure Plane

The scope is given structure on the software side through interaction design, in which we define how the system behaves in response to the user. For information spaces, the structure is the information architecture: the arrangement of content elements within the information space.

The Skeleton Plane

The skeleton plane breaks down into three components. On both sides, we must address information design: the presentation of information in a way that facilitates understanding. For software products, the skeleton also includes interface design, or arranging interface elements to enable users to interact with the functionality of the system. The interface for an information space is its navigation design: the set of screen elements that allow the user to move through the information architecture.

The Surface Plane

Finally, we have the surface. Regardless of whether we are dealing with a software product or an information space, our concern here is the same: the visual design, or the look of the finished product.

Using the Elements

Few sites fall exclusively on one side of this model or the other. Within each plane, the elements must work together to accomplish that plane's goals. For example, information design, navigation design, and interface design jointly define the skeleton of a site. The

© Copyright Virtual University of Pakistan

effects of decisions you make about one element from all other elements on the plane is very difficult. All the elements on every plane have a common
Function-in this example, defining the sites skeleton-even if they perform that function in different ways. Elements on the plane are very difficult. All the elements on every plane have a common function-in this example; defining the site's skeleton-even if they perform that function in different ways.
This model, divided up into neat boxes and planes, is a convenient way to think about user experience problems. In reality, however, the lines between these areas are not so clearly drawn. Frequently, it can be difficult to identify whether a particular user experience problem is best solved through attention to one element instead of another. Can a change to the visual design do the trick, or will the underlying navigation design have to be reworked? Some problems require attention in several areas at once, and some seem to straddle the borders identified in this model.
The way organizations often delegate responsibility for user experience issues only complicates matters further. In some organizations, you will encounter people with job titles like information architect or interface designer. Don't be confused by this. These people generally have expertise spanning many of the elements of user experience, not just the specialty indicated by their title. It's not necessary for thinking about each of these issues.
A couple of additional factors go into shaping the final user experience that you won't find covered in detail here. The first of these is content. The old saying (well, old in Web years) is that "content is king" on the Web. This is absolutely true-the single most important thing most Web sites can offer to their users is content that those users will find valuable.

© Copyright Virtual University of Pakistan

Lecture 36.

Behavior & Form – Part IV

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the significance of undo
- Discuss file and save

36.1 Understanding undo

Undo is the remarkable facility that lets us reverse a previous action. Simple and elegant, the feature is of obvious value. Yet, when we examine undo from a goal-directed point of view, there appears to be a considerable variation in purpose and method. Undo is critically important for users, and it's not quite as simple as one might think. This lecture explores the different ways users think about undo and the different uses for such a facility.

Users and Undo

Undo is the facility traditionally thought of as the rescuer of users in distress; the knight in shining armor; the cavalry galloping over the ridge; the superhero swooping in at the last second.

As a computational facility, undo has no merit. Mistake-free as they are, computers have no need for undo. Human beings, on the other hand, make mistakes all the time, and undo is a facility that exists for their exclusive use. This singular observation should immediately tell us that of all the facilities in a program, undo should be modeled the least like its construction methods—its implementation model —and the most like the user's mental model.

Not only do humans make mistakes, they make mistakes as part of their everyday behavior. From the standpoint of a computer, a false start, a misdirected glance, a pause, a hiccup, a sneeze, a cough, a blink, a laugh, an "uh," a "you know" are all errors. But from the standpoint of the human user, they are perfectly normal. Human mistakes are so commonplace that if you think of them as "errors" or even as abnormal behavior, you will adversely affect the design of your software.

User mental models of mistakes

Users don't believe, or at least don't want to believe, that they make mistakes. This is another way of saying that the user's mental model doesn't typically include error on his part. Following the user's mental model means absolving the user of blame. The implementation model, however, is based on an error-free CPU. Following the implementation model means proposing that all culpability must rest with the user. Thus, most software assumes that it is blameless, and any problems are purely the fault of the user.

The solution is for the user interface designer to completely abandon the idea that the user can make a mistake — meaning that everything the user does is something he or she

© Copyright Virtual University of Pakistan

considers to be valid and reasonable. Users don't like to admit to mistakes in their own minds, so the program shouldn't contradict these actions in its interactions with users.

Undo enables exploration

If we design software from the point of view that nothing users do should constitute a mistake, we immediately begin to see things differently. We cease to imagine the user as a module of code or a peripheral that drives the computer, and we begin to imagine
him as an explorer, probing the unknown. We understand that exploration involves inevitable forays into blind alleys and box canyons, down dead ends and into dry holes. It is natural for humans to experiment, to vary their actions, to probe gently against the veil of the unknown to see where their boundaries lie. How can they know what they can do with a tool unless they experiment with it? Of course the degree of willingness to experiment varies widely from person to person, but most people experimental least a little bit.

Programmers, who are highly paid to think like computers (Cooper, 1999), view such behavior only as errors that must be handled by the code. From the implementation model — necessarily the programmer's point of view such gentle, innocent probing represents a continuous series of "mistakes". From our more-enlightened, mental model point-of-view, these actions are natural and normal. The program has the choice of either rebuffing those perceived mistakes or assisting the user in his explorations. Undo is thus the primary tool for supporting exploration in software user interfaces. It allows the user to reverse one or more previous actions if he decides to change his mind.

A significant benefit of undo is purely psychological: It reassures users. It is much easier to enter a cave if you are confident that you can get back out of it at any time. The undo function is that comforting rope ladder to the surface, supporting the user's willingness to explore further by assuring him that he can back out of any dead-end caverns.

Curiously, users often don't think about undo until they need it, in much the same way that homeowners don't think about their insurance policies until a disaster strikes. Users frequently charge into the cave half prepared, and only start looking for the rope ladder — for undo — after they have encountered trouble

Designing an Undo Facility

Although users need undo, it doesn't directly support a particular goal they bring to their tasks. Rather, it supports a necessary condition — trustworthiness — on the way to a real goal. It doesn't contribute positively to attaining the user's goal, but keeps negative occurrences from spoiling the effort.

Users visualize the undo facility in many different ways depending on the situation and their expectations. If the user is very computer-naive, he might see it as an unconditional panic button for extricating himself from a hopelessly tangled misadventure. A more experienced computer user might visualize undo as a storage facility for deleted data. A really computer-sympathetic user with a logical mind might see it as a stack of procedures that can be undone one at a time in reverse order. In order to create an effective undo facility, we must satisfy as many of these mental models as we expect our users will bring to bear.

The secret to designing a successful undo system is to make sure that it supports typically used tools and avoids any hint that undo signals (whether visually, audibly, or textually) a failure by the user. It should be less a tool for reversing errors and more one for supporting exploration. Errors are generally single, incorrect actions.

© Copyright Virtual University of Pakistan

Exploration, by contrast, is a long series of probes and steps, some of which are keepers and others that must be abandoned.
Undo works best as a global, program-wide function that undoes the last action regardless of whether it was done by direct manipulation or through a dialog box. This can make undo problematic for embedded objects. If the user makes changes to a spreadsheet embedded in a Word document, clicks on the Word document, and then invokes undo, the most recent Word action is undone instead of the most recent spreadsheet action. Users have a difficult time with this. It fails to render the juncture between the spreadsheet and the word-processing document seamlessly: The undo function ceases to be global and becomes modal. This is not an undo problem per se, but a problem with the embedding technology.

36.2 Types and Variants of

As is so common in the software industry, there is no adequate terminology to describe the types of undo that exist — they are uniformly called undo and left at that. This language gap contributes to the lack of innovation in new and better variants of undo. In this section, we define several undo variants and explain their differences.

36.3 Incremental and procedural actions

First, consider what objects undo operates on: the user's actions. A typical user action in a typical application has a procedure component—what the user did — and an optional data component — what information was affected. When the user requests an undo function, the procedure component of the action is reversed, and if the action had an optional data component — the user added or deleted data—that data will be deleted or added back, as appropriate. Cutting, pasting, drawing, typing, and deleting are all actions that have a data component, so undoing them involves removing or replacing the affected text or image parts. Those actions that include a data component are ailed incremental actions.
Many undoable actions are data-free transformations such as a paragraph reformatting operation in a word processor or a rotation in a drawing program. Both these operations act on data but neither of them add or delete data. Actions like these (with only a procedure component) are procedural actions. Most existing undo functions don't discriminate between procedural and incremental actions but simply reverse the most recent action.

Blind and explanatory undo

Normally, undo is invoked by a menu item or toolbar control with an unchanging label or icon. The user knows that triggering the idiom undoes the last operation, but there is no indication of what that operation is. This is called a blind undo. On the other hand, if the idiom includes a textual or visual description of the particular operation that will be undone it is an explanatory undo. If, for example, the user's last operation was to type in the word design, the undo function on the menu says Undo Typing design. Explanatory undo is, generally, a much more pleasant feature than
blind undo. It is fairly easy to put on a menu item, but more difficult to put on a toolbar control, although putting the explanation in a ToolTip is a good compromise.

36.4 Single and multiple undo

The two most-familiar types of undo in common use today are single undo and multiple undo. Single undo is the most basic variant, reversing the effects of the most recent user

© Copyright Virtual University of Pakistan

action, whether procedural or incremental. Performing a single undo twice usually undoes the undo, and brings the system back to the state it was in before the first undo was activated.
This facility is very effective because it is so simple to operate. The user interface is simple and clear, easy to describe and remember. The user gets precisely one free lunch. This is by far the most frequently implemented undo, and it is certainly adequate, if not optimal, for many programs. For some users, the absence of this simple undo is sufficient grounds to abandon a product entirely.
The user generally notices most of his command mistakes right away: Something about what he did doesn't feel or look right, so he pauses to evaluate the situation. If the representation is clear, he sees his mistake and selects the undo function to set things back to the previously correct state; then he proceeds.
Multiple undo can be performed repeatedly in succession — it can revert more than one previous operation, in reverse temporal order. Any program with simple undo must remember the user's last operation and, if applicable, cache any changed data. If the program implements multiple undo, it must maintain a stack of operations, the depth of which may be settable by the user as an advanced preference. Each time undo is invoked, it performs an incremental undo; it reverses the most recent operation, replacing or removing data as necessary and discarding the restored operation from the stack.

LIMITATIONS OF SINGLE UNDO

The biggest limitation of single-level, functional undo is when the user accidentally short-circuits the capability of the undo facility to rescue him. This problem crops up when the user doesn't notice his mistake immediately. For example, assume he deletes six paragraphs of text, then deletes one word, and then decides that the six paragraphs were erroneously deleted and should be replaced. Unfortunately, performing undo now merely brings back the one word, and the six paragraphs are lost forever. The undo function has failed him by behaving literally rather than practically. Anybody can clearly see that the six paragraphs are more important than the single word, yet the program freely discarded those paragraphs in favor of the one word. The program's blindness caused it to keep a quarter and throw away a fifty-dollar bill, simply because the quarter was offered last.
In some programs any click of the mouse, however innocent of function it might be, causes the single undo function to forget the last meaningful thing the user did. Although multiple undo solves these problems, it introduces some significant problems of its own.

LIMITATIONS OF MULTIPLE UNDO

"The response to the weaknesses of single-level undo has been to create a multiple-level implementation of the same, incremental undo. The program saves each action the user takes. By selecting undo repeatedly, each action can be undone in the reverse order of its original invocation. In the above scenario, the user can restore the deleted word with the first invocation of undo and restore the precious six paragraphs with a second invocation. Having to redundantly re-delete the single word is a small price to pay for being able to recover those six valuable paragraphs. The excise of the one-word re-deletion tends to not be noticed, just as we don't notice the cost of ambulance trips: Don't quibble over the little stuff when lives are at stake. But this doesn't change the fact that the undo mechanism is built on a faulty model, and in other circumstances, undoing functions in a strict LIFO (Last In. First Out) order can make the cure as painful as the disease.

© Copyright Virtual University of Pakistan

Imagine again our user deleting six paragraphs of text, then calling up another document and performing a global find-and-replace function. In order to retrieve the missing six paragraphs, the user must first unnecessarily undo the rather complex global find-and replace operation. This time, the intervening operation was not the insignificant single-word deletion of the earlier exam-pie. The intervening operation was complex and difficult and having to undo it is clearly an unpleasant, excise effort. It would sure be nice to be able to choose which operation in the queue to undo and to be able to leave intervening — but valid — operations untouched.

THE MODEL PROBLEMS OF MULTIPLE UNDO

The problems with multiple undo are not due to its behavior as much as they are due to its manifest model. Most undo facilities are constructed in an unrelentingly function-centric manner. They remember what the user does function-by-function and separate the user's actions by individual function. In the time-honored way of creating manifest models that follow implementation models, undo systems tend to model code and data structures instead of user goals. Each click of the Undo button reverses precisely one function-sized bite of behavior. Reversing on a function-by-function basis is a very appropriate mental model for solving most simple problems caused by the user making an erroneous entry. Users sense it right away and fix it right away, usually within a two- or three-function limit. The Paint program in Windows 95, for example, had a fixed, three-action undo limit. However, when the problem grows more convoluted, the incremental, multiple undo models don’t scale up very well.

TOU BET YOUR LIFO

When the user goes down a logical dead-end (rather than merely mistyping data), he can often proceed several complex steps into the unknown before realizing that he is lost and needs to get a bearing on known territory. At this point, however, he may have performed several interlaced functions, only some of which are undesirable. He may well want to keep some actions and nullify others, not necessarily in strict reverse order. What if the user entered some text, edited it, and then decided to undo the entry of that text but not undo the editing of it? Such an operation is problematic to implement and explain. Neil Rubenking offers this pernicious example: Suppose the user did a global replace changing *tragedy* to *catastrophe* and then another changing *cat* to *dog*. To undo the first without undoing the second, can the program reliably fix all the *dog strophes?*

In this more complex situation, the simplistic representation of the undo as a single, straight-line, LIFO stack doesn't satisfy the way it does in-simpler situations. The user may be interested in studying his actions as a menu and choosing a discontinuous subset of them for reversion, while keeping some others. These demands an explanatory undo with a more robust presentation than might otherwise be necessary for a normal, blind, multiple undo. Additionally, the means for selecting from that presentation must be more sophisticated. Representing the operation in the to clearly show the user what he is actually undoing is a more difficult problem.

Redo

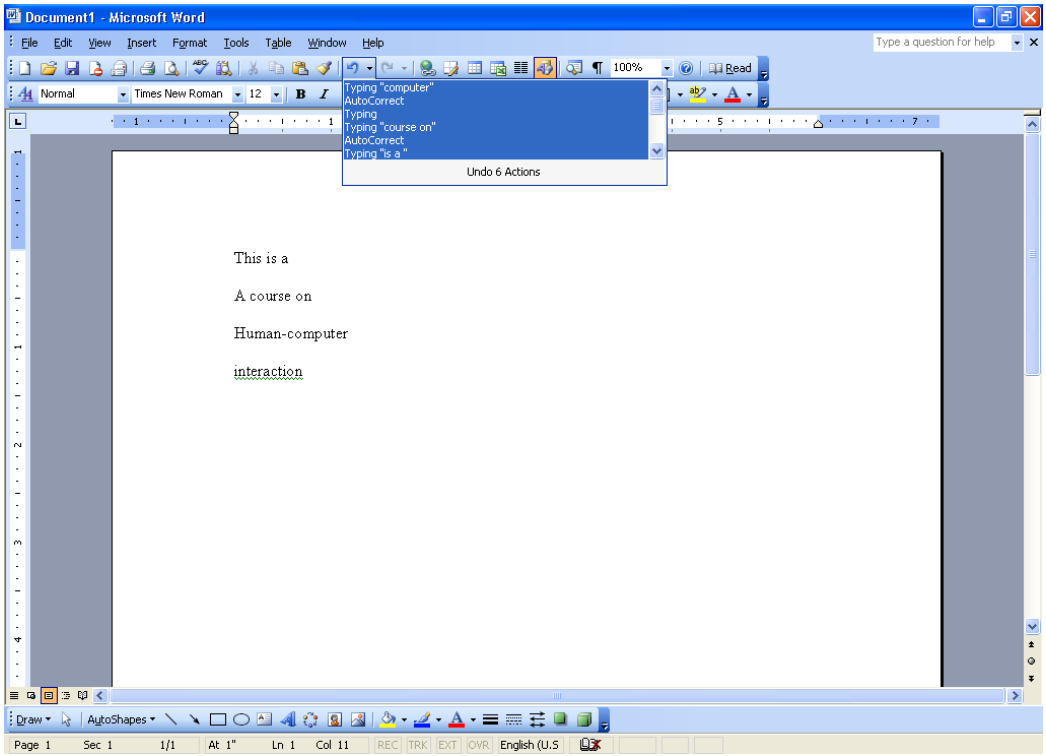
The redo function came into being as the result of the implementation model for undo, wherein operations must be undone in reverse sequence, and in which no operation may be undone without first undoing all of the valid intervening operations. Redo essentially undoes the undo and is easy to implement if the programmer has already gone to the effort to implement undo.

© Copyright Virtual University of Pakistan

Redo avoids a diabolical situation in multiple undo. If the user wants to back out of half-dozen or so operations, he clicks the Undo control a few times, waiting to see things return to the desired state. It is very easy in this situation to press Undo one time too many. He immediately sees that he has undone something desirable. Redo solves this problem by allowing him to undo the undo, putting back the last good action.
Many programs that implement single undo treat the last undone action as an undoable action. In effect, this makes a second invocation of the undo function a minimal redo function.

Group multiple undo

Microsoft Word has an unusual undo facility, a variation of multiple undo we will call group multiple undo. It is multiple-level, showing a textual description of each operation in the undo stack. You can examine the list of past operations and select some operation in the list to undo; however, you are not undoing that one operation, but rather all operations back to that point, inclusive (see Figure on next page).
Essentially, you cannot recover the six missing paragraphs without first reversing all the intervening operations. After you select one or more operations to undo, the list of undone operations is now available in reverse order in the Redo control. Redo works exactly the same way as undo works. You can select as many operations to redo as desired and all operations up to that specific one will be redone.



The program offers two visual cues to this fact. If the user selects the fifth item in the list, that item and all four items previous to it in the list are selected. Also, the text legend says "Undo 5 actions." The fact that the designers had to add that text legend tells me that, regardless of how the programmers constructed it, the users were applying a different mental model. The users imagined that they could go down the list and select a single action from the past to undo. The program didn't offer that option, so the signs were posted.

© Copyright Virtual University of Pakistan

This is like the door with a pull handle pasted with Push signs — which everybody still pulls on anyway.

36.5 Other Models for Undo-Like Behavior

The manifest model of undo in its simplest form, single undo, conforms to the user's mental model: "I just did something I now wish I hadn't done. I want to click a button and undo that last thing I did "Unfortunately, this manifest model rapidly diverges from the user's mental model as the complexity of the situation grows. In this section, we discuss models of undo-like behavior that work a bit differently from the more standard undo and redo idioms.

Comparison: What would this look like?

Resides providing robust support for the terminally indecisive, the paired undo-redo function is a convenient comparison tool. Say you'd like to compare the visual effect of ragged-right margins against justified right margins. Beginning with ragged-right, you invoke Justification. Now you click Undo to see ragged-right and now you press Redo to see justified margins again. In effect, toggling between Undo and Redo implements a comparison or what-if function; it just happens to be represented in the form of its implementation model. If this same
function were to be added to the interface following the user's mental model, it might be manifested as a comparison control. This function would let you repeatedly take one step forward or backward to visually compare two states.

Some Sony TV remote controls include a function labeled Jump, which switches between the current channel and the previous channel—very convenient for viewing two programs concurrently. The jump function provides the same usefulness as the undo-redo function pair with a single command—a 50% reduction in excise for the same functionality.

When used as comparison functions, undo and redo are really one function and not two. One says "Apply this change," and the other says "Don't apply this change." A single Compare button might more accurately represent the action to the user. Although we have been describing this tool in the context of a text-oriented word processing program, a compare function might be most useful in a graphic manipulation or drawing program, where the user is applying successive visual transformations on images. The ability to see the image with the transformation quickly and easily compared to the image without the transformation would be a great help to the digital artist.

Doubtlessly, the compare function would remain an advanced function. Just as the jump function is probably not used by a majority of TV users, the Compare button would remain one of those niceties for frequent users. This shouldn't detract from its usefulness, however, because drawing programs tend to be used very frequently by those who use them at all. For programs like this, catering to the frequent user is a reasonable design choice.

Category-specific Undo

The Backspace key is really an undo function, albeit a special one. When the user mistypes, the Backspace key "undoes" the erroneous characters. If the user mistypes something, then enters an unrelated function such as paragraph reformatting, then presses the Backspace key repeatedly, the mistyped characters are erased and the reformatting operation is ignored. Depending on how you look at it, this can be a great flexible advantage giving the user the ability to undo discontiguously at any selected location. You could also see it as a trap for the user because he can move the cursor and inadvertently backspace away characters that were not the last ones keyed in.

Logic says that this latter case is a problem. Empirical observation says that it is rarely a problem for users. Such discontiguous, incremental undo — so hard to explain in words — is so natural and easy to actually use because everything is visible: The user can clearly see

© Copyright Virtual University of Pakistan

what will be backspaced away. Backspace is a classic example of an incremental undo, reversing only some data while ignoring other, intervening actions. Yet if you imagined an undo facility that had a pointer that could be moved and that could undo the last function that occurred where the pointer points, you'd probably think that such a feature would be patently unmanageable and would confuse the bejabbers out of a typical user. Experience tells us that Backspace does nothing of the sort. It works as well as it does because its behavior is consistent with the user's mental model of the cursor: Because it is the source of added characters, it can also reasonably be the locus of deleted characters.

Using this same knowledge, we could create different categories of incremental undo, like a format-undo function that would only undo preceding format commands and other types of category-specific undo actions. If the user entered some text, changed it to italic, entered some more text, increased the paragraph indentation, entered some more text, then pressed the Format-Undo key, only the indentation increase would be undone. A second press of the Format-Undo key would reverse the italicize operation. But neither invocation of the format-undo would affect the content.

What are the implications of category-specific undo in a non-text program? In a graphics drawing program, for example, there could be separate undo commands for pigment application tools, transformations, and cut-and-paste. There is really no reason why we couldn't have independent undo functions for each particular class of operation in a program.

Pigment application tools include all drawing implements — pencils, pens, fills, sprayers, brushes — and all shape tools — rectangles, lines, ellipses, arrows. Transformations include all image-manipulation tools — shear, sharpness, hue, rotate, contrast, line weight. Cut-and-paste tools include all lassos, marquees, clones, drags, and other repositioning tools. Unlike the Backspace function in the word processor, undoing a pigment application in a draw program would be temporal and would work independently of selection. That is, the pigment that is removed first would be the last pigment applied, regardless of the current selection. In text, there is an implied order from the upper-left to the lower-right. Deleting from the lower-right to the upper-left maps to a strong, intrinsic mental model; so it seems natural. In a drawing, no such conventional order exists so any deletion order other than one based on entry sequence would be disconcerting to the user.

A better alternative might be to undo within the current selection only. The user selects a graphic object, for example, and requests a transformation-undo. The last transformation to have been applied to that *selected object* would be reversed.

Most software users are familiar with the incremental undo and would find a category-specific undo novel and possibly disturbing. However, the ubiquitousness of the Backspace key shows that incremental undo is a learned behavior that users find to be helpful. If more programs had modal undo tools, users would soon adapt to them. They would even come to expect them the way they expect to find the Backspace key on word processors.

Deleted data buffers

As the user works on a document for an extended time, his desire for a repository of deleted text grows, it is not that he finds the ability to incrementally undo commands useless but rather that reversing actions can cease to be so function-specific. Take for example, our six missing paragraphs. If they are separated from us by a dozen complex formatting commands, they can be as difficult to reclaim by undo as they are to re-key. The user is

© Copyright Virtual University of Pakistan

thinking, "If the program would just remember the stuff I deleted and keep it in a special place, 1 could go get what I want directly."

What the user is imagining is a repository of the data components of his actions, rather than merely a LIFO stack of procedural — a deleted data buffer. The user wants the missing text with out regard to which function elided it. The usual manifest model forces him not only to be aware of every intermediate step but to reverse each of them, in turn. To create a facility more amenable to the user, we can create, in addition to the normal undo stack, an independent buffer that collects all deleted text or data. At any time, the user can open this buffer as a document and use standard cut-and-paste or click-and-drag idioms to examine and recover the desired text. If the entries in this deletion buffer are headed with simple date stamps and document names, navigation would be very simple and visual

The user could browse the buffer of deleted data at will, randomly, rather than sequentially. Finding those six missing paragraphs would be a simple, visual procedure, regardless of the number or type of complex, intervening steps he had taken. A deleted data buffer should be offered in addition to the regular, incremental, multiple undo because it complements it. The data must be saved in a buffer, anyway. This feature would be quite useful in all programs, too, whether spreadsheet, drawing program, or invoice generator.

Mile stoning and reversion

Users occasionally want to back up long distances, but when they do, the granular actions are not terrifically important. The need for an incremental undo remains, but discerning the individual components of more than the last few operations is overkill in most cases. Milestoning, simply makes a copy of the entire document the way a camera snapshot freezes an image in time. Because milestoning involves the entire document, it is always implemented by direct use of the file system. The biggest difference between milestoning and other undo systems is that the user must explicitly request the milestone — recording a copy or snapshot of the document. After he has done this, he can proceed to safely modify the original. If he later decides that his changes were undesirable, he can return to the saved copy—a previous version of the document.

Many tools exist to support the milestoning concept in source code; but as yet, no programs the authors are aware of present it directly to the user. Instead, they rely on the file system's interface, which, as we have seen, is difficult for many users to understand. If milestoning were rendered in a non-file-system user model, implementation would be quite easy, and its management would be equally simple. A single button control could save the document in its current state. The user could save as many versions at any interval as he desires. To return to a previously milestone version, the user would access a reversion facility.

The reversion facility is extremely simple — too simple, perhaps. Its menu item merely says, Revert to Milestone. This is sufficient for a discussion of the file system; but when considered as part of an undo function, it should offer more information. For example, it should display a list of the available saved versions of that document along with some information about each one, such as the time and day it was recorded, the name of the person who recorded it. the size, and some optional user-entered notes. The user could choose one of these versions, and the program would load it, discarding any intervening changes.

© Copyright Virtual University of Pakistan

Freezing

Freezing, the opposite of mile stoning, involves locking the data in a document so that it cannot be changed. Anything that has been entered becomes un-modifiable, although new data can be added. Existing paragraphs are untouchable, but new ones can be added between older ones.
This method is much more useful for a graphic document than for a text document. It is much like an artist spraying a drawing with fixative. All marks made up to that point are now permanent, yet new marks can be made at will. Images already placed on the screen are locked down and cannot be changed, but new images can be freely superimposed on the older ones. Procreate Painter offers a similar feature with its Wet Paint and Dry Paint commands.

Undo Proof Operation

Some operations simply cannot be undone because they involve some action that triggers a device « not under the direct control of the program. After an e-mail message has been sent, for example, ^ there is no undoing it. After a computer has been turned off without saving data, there is no undoing the loss. Many operations, however, masquerade as undo-proof, but they are really easily reversible. For example, when you save a document for the first time in most programs, you can choose a name for the file. But almost no program lets you rename that file. Sure, you can Save As under another name, but that just makes *another* file under the new name, leaving the old file untouched under the old name. Why isn't a filename undo provided? Because it doesn't fall into ^ the traditional view of what undo is for, programmers generally don't provide a true undo function for changing a filename. Spend some time looking at your own application and see if you can find functions that seem as if they should be undoable, but currently aren't. You may be surprised by how many you find.

36.6 Rethinking Files and Save

If you have ever tried to teach your mom how to use a computer, you will know that *difficult* doesn't really do the problem justice. Things start out ail right: Start up the word processor and key in a letter. She's with you all the way. When you are finally done, you click the Close button, and up pops a dialog box asking "Do you want to save changes?" You and Mom hit the wall together. She looks at you and asks, "What does this mean: “Is everything okay?”
The part of modern computer systems that is the most difficult for users to understand is the file system, the facility that stores programs and data files on disk. Telling the uninitiated about disks is very difficult. The difference between main memory and disk storage is not clear to most people. Unfortunately, the way we design our software forces users — even your mom — to know the difference. This chapter provides a different way of presenting interactions involving files and disks — one that is more in harmony with the mental models of our users.

What's Wrong with Saving Changes to Files?

Every program exists in two places at once: in memory and on disk. The same is true of every file. However, most users never truly grasp the difference between memory and disk storage and how it pertains to the tasks they perform on documents in a computer system. Without a doubt, the file system — along with the disk storage facility it manages — is the primary cause of disaffection with computers among non-computer-professionals.

© Copyright Virtual University of Pakistan

When that Save Changes? dialog box, shown in Figure, opens users suppress a twinge of fear and confusion and click the Yes button out of habit. A dialog box that is always answered the same way is a redundant dialog box that should be eliminated.

The Save Changes dialog box is based on a poor assumption: that saving and not saving are equally probable behaviors. The dialog gives equal weight to these two options even though the Yes button is clicked orders of magnitude more frequently than the No button. As discussed in Chapter 9, this is a case of confusing possibility and probability. The user *might* say no, but the user *will almost always* say yes. Mom is thinking, "If 1 didn't want those changes, why would I have closed the document with them in there?" To her, the question is absurd. There's something else a bit odd about this dialog: Why does it only ask about saving changes when you are all done? Why didn't it ask when you actually made them? The connection between closing a document and saving changes isn't all that natural, even though power users have gotten quite familiar with it.

The program issues the Save Changes dialog box when the user requests Close or Quit because that is the time when it has to reconcile the differences between the copy of the document in memory and the copy on the disk. The way the technology actually implements the facility associates saving changes with Close and Quit, but the user sees no connection. When we leave a room, we don't consider discarding all the changes we made while we were there. When we put a book back on the shelf, we don't first erase any comments we wrote in the margins.

As experienced users, we have learned to use this dialog box for purposes for which it was never intended. There is no easy way to undo massive changes, so we use the Save Changes dialog by choosing No. If you discover yourself making big changes to the wrong file, you use this dialog as a kind of escape valve to return things to the status quo. This is handy, but it's also a hack: There are better ways to address these problems (such as an obvious Revert function). So what is the real problem? The file systems on modern personal computer operating systems, like Windows XP or Mac OS X, are technically excellent. The problem Mom is having stems from the simple mistake of faithfully rendering that excellent implementation model as an interface for users.

Problems with the Implementation Model

The computer's file system is the tool it uses to manage data and programs stored on disk. This means the large hard disks where most of your information resides, but it also includes your floppy disks, ZIP disks, CD-ROMs, and DVDs if you have them. The Finder on the Mac and the Explorer in Windows graphically represent the file system in all its glory.

Even though the file system is an internal facility that shouldn't — by all rights — affect the user, it creates a large problem because the influence of the file system on the interface of most programs is very pervasive. Some of the most difficult problems facing interaction designers concern the file system and its demands. It affects our menus, our dialogs, even the procedural framework of our programs; and this influence is likely to continue indefinitely unless we make a concerted effort to stop it.

Currently, most software treats the file system in much the same way that the operating system shell does (Explorer. Kinder). This is tantamount to making you deal with your car in the same way a mechanic does. Although this approach is unfortunate from an interaction perspective, it is a de facto standard, and there is considerable resistance to improving it,

Closing and unwanted changes

We computer geeks are conditioned to think that Close is the time and place for abandoning unwanted changes if we make some error or are just noodling around. This is not correct because the proper time to reject changes is when the changes are made. We even

© Copyright Virtual University of Pakistan

have a well-established idiom to support this: The Undo function is the proper facility for eradicating changes.

Save As

When you answer yes to the Save Changes dialog, many programs then present you with the Save As dialog box.

Most users don't understand the concept of manual saving very well, so from their point of view, the existing name of this dialog box doesn't make much sense. Functionally, this dialog offers two things: It lets users name a file, and it lets them choose which directory to place it in. Both of these functions demand intimate knowledge of the file system. The user must know how to formulate a filename and how to navigate through the file directory. Many users who have mastered the name portion have completely given up on understanding the directory tree. They put their documents in the directory that the program chooses for a default. All their files are stored in a single directory. Occasionally, some action will cause the program to forget its default directory, and these users must call in an expert to find their files for them.

The Save As dialog needs to decide what its purpose truly is. If it is to name and place files, then it does a very poor job. After the user has named and placed a file, he cannot then change its name or its directory — at least not with this dialog, which purports to offer naming and placing functions — nor can he with any other tool in the application itself. In fact, in Windows XP, you can rename other files using this dialog, hut not the ones you are currently working on. Huh? The idea, one supposes, is to allow you to rename other previously saved milestones of your document because you can't rename the current one. But both operations ought to be possible and be allowed.

Beginners are out of luck, but experienced users learn the hard way that they can close the document, change to the Explorer, rename the file, return to the application, summon the Open dialog from the File menu, and reopen the document. In case you were wondering, the Open dialog doesn't allow renaming or repositioning either, except in the bizarre cases mentioned in the previous paragraph.

Forcing the user to go to the Explorer to rename the document is a minor hardship, but therein lies a hidden trap. The bait is that Windows easily supports several applications running simultaneously. Attracted by this feature, the user tries to rename the file in the Explorer without first closing the document in the application. This very reasonable action triggers the trap, and the steel jaws clamp down hard on his leg. He is rebuffed with a rude error message box. He didn't first close the document — how would he know? Trying to rename an open file is a sharing violation, and the operating system rejects it with a patronizing error message box.

The innocent user is merely trying to rename his document, and he finds himself lost in an , archipelago of operating system arcana. Ironically, the one entity that has both the authority and the responsibility to change the document's name while it is still open, the application itself, refuses to even try.

36.7 Archiving

There is no explicit function for making a copy of. or archiving, a document. The user must accomplish this with the Save As dialog, and doing so is as clear as mud. Even if there were a Copy * command, users visualize this function differently. If we are working, for example, on a document called Alpha, some people imagine that we would create a file called Copy of Alpha and store that away. Others imagine that we put Alpha away and continue work on Copy of Alpha.

© Copyright Virtual University of Pakistan

The latter option will likely only occur to those who are already experienced with the implementation model of file systems. It is how we do it today with the Save As dialog: You have already saved the file as Alpha; and then you explicitly call up the Save As dialog and change the name. Alpha is closed and put away on disk, and Copy of Alpha is left open for editing. This action makes very little sense from a single-document viewpoint of the world, and it also offers a really nasty trap for the user

Here is the completely reasonable scenario that leads to trouble: Let's say that you have been editing Alpha for the last twenty minutes and now wish to make an archival copy of it on disk so you can make some big but experimental changes to the original. You call'up the Save As dialog box and change the file name to New Alpha. The program puts Alpha away on disk leaving you to edit New Alpha. But Alpha was never saved, so it gets written to disk without any of the changes you made in the last twenty minutes! Those changes only exist in the New Alpha copy that is currently in memory — in the program. As you begin cutting and pasting in New Alpha, trusting that your handiwork is backed up by Alpha, you are actually modifying the sole copy of this information.

Everybody knows that you can use a hammer to drive a screw or pliers to bash in a nail, but any skilled craftsperson knows that using the wrong too! for the job will eventually catch up with you. The tool will break or the work will be hopelessly ruined. The Save As dialog is the wrong tool for making and managing copies, and it is the user who will eventually have to pick up the pieces.

36.8 Implementation Model versus Mental Model

The implementation model of the file system runs contrary to the mental model almost all users bring to it. Most users picture electronic files like printed documents in the real world, and they imbue them with the behavioral characteristics of those real objects. In the simplest terms, users visualize two salient facts about all documents: First, there is only one document; and second, it belongs to them. The file system's implementation model violates both these rules: There are always two copies of the document, and they both belong to the program.

Every data file, every document, and every program, while in use by the computer, exists in two places at once: on disk and in main memory. The user, however, imagines his document as a book on a shelf. Let's say it is a journal. Occasionally, it comes down off the shelf to have something added to it. There is only one journal, and it either resides on the shelf or it resides in the user's hands. On the computer, the disk drive is the shelf, and main memory is the place where editing takes place, equivalent to the user's hands. But in the computer world, the journal doesn't come off the shelf. Instead a copy is made, and that *copy* is what resides in computer memory. As the user makes changes to the document, he is actually making changes to the copy in memory, while the original remains untouched on disk. When the user is done and closes the document, the pro gram is faced with a decision: whether to replace the original on disk with the changed copy from memory, or to discard the altered copy. Prom the programmer's point of view, equally concerned with all possibilities, this choice could go either way. From the software's implementation model point of view, the choice is the same either way. However, from the user's point of view, there is no decision to be made at all. He made his changes, and now he is just putting the document away. If this were happening with a paper journal in the physical world, the user would have pulled it off the shelf, penciled in some additions, and then replaced it on the shelf. It's as if the shelf suddenly were to speak up, asking him if he really wants to keep those changes

© Copyright Virtual University of Pakistan

36.9 Dispensing with the Implementation Model of the File System

Right now, serious programmer-type readers are beginning to squirm in their seats. They are thinking that we're treading on holy ground: A pristine copy on disk is a wonderful thing, and we'd better not advocate getting rid of it. Relax! There is nothing wrong with our file systems. We simply need to hide its existence from the user. We can still offer to him all the advantages of that extra copy on disk without exploding his mental model.

If we begin to render the file system according to the user's mental model we achieve a significant advantage: We can all teach our moms how to use computers. We won't have to answer her pointed questions about the inexplicable behavior of the interface. We can show her the program and explain how it allows her to work on the document; and, upon completion, she can store the document on the disk as though it were a journal on a shelf. Our sensible explanation won't be interrupted by that Save Changes? dialog. And Mom is representative of the mass-market of computer buyers.

Another big advantage is that interaction designers won't have to incorporate clumsy file system awareness into their products. We can structure the commands in our programs according to the goals of the user instead of according to the needs of the operating system. We no longer need to call the left-most menu the File menu. This nomenclature is a bold reminder of how technology currently pokes through the facade of our programs. Changing the name and contents of the File menu violates an established, though unofficial, standard. But the benefits will far outweigh any dislocation the change might cause. There will certainly be an initial cost as experienced users get used to the new presentation, but it will be far less than you might suppose. This is because these power users have already shown their ability and tolerance by learning the implementation model. For them, learning the better model will be no problem, and there will be no loss of functionality for them. The advantage for new users will be immediate and significant. We computer professionals forget how tall the mountain is after we've climbed it. but everyday newcomers approach the base of this Everest of computer literacy and are severely discouraged. Anything we can do to lower the heights they must scale will make a big difference, and this step will tame some of the most perilous peaks.

36.10 Designing a Unified File Presentation Model

Properly designed software will always treat documents as single instances, never as a copy on disk and a copy in memory: a unified file model. It's the file system's job to manage information not in main memory, and it does so by maintaining a second copy on disk. This method is correct, but it is an implementation detail that only confuses the user. Application software should conspire with the file system to hide this unsettling detail from the user. If the file system is going to show the user a file that cannot be changed because it is in use by another program, the file system should indicate this to the user. Showing the filename in red or with a special symbol next to it would be sufficient. A new user might still get an error message as shown in Figure 33-3; but, at least, some visual clues would be present to show him that there was a *reason* why that error cropped up.

Not only are there two copies of all data files in the current model, but when they are running, there are two copies of all programs. When the user goes to the Taskbar's Start menu and launches his word processor, a button corresponding to Word appears on the Taskbar. But if he returns to the Start menu. Word is still there! Prom the users point of view, he has pulled his hammer out of his toolbox only to find that there is still a hammer in his toolbox!

© Copyright Virtual University of Pakistan

This should probably not be changed; after all, one of the strengths of the computer is its capability to have multiple copies of software running simultaneously. But the software should help the user to understand this very non-intuitive action. The Start menu could, for example make some reference to the already running program.

© Copyright Virtual University of Pakistan

Lecture 37.

Behavior & Form - Part V

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Discuss Unified Document Management
- Understand considerate and smart software

37.1 Unified Document Management

The established standard suite of file management for most applications consists of the Save As dialog, the Save Changes dialog, and the Open File dialog. Collectively, these dialogs are, as we've shown, confusing for some tasks and completely incapable of performing others. The following is a different approach that manages documents according to the user's mental model.

Besides rendering the document as a single entity, there are several goal-directed functions that the user may have need for and each one should have its own corresponding function.

- Automatically saving the document
- Creating a copy of the document
- Creating a milestone/milestoned copy of the document
- Naming and renaming the document
- Placing and repositioning the document
- Specifying the stored format of the document
- Reversing some changes
- Abandoning all changes

Automatically saving the document

One of the most important functions every computer user must learn is how to save a document. Invoking this function means taking whatever changes the user has made to the copy in computer memory and writing them onto the disk copy of the document. In the unified model, we abolish all user interface recognition of the two copies, so the Save function disappears completely from the mainstream interface. That *doesn't* mean that it disappears from the program; it is still a very necessary operation.

The program should *automatically* save the document. At the very least, when the user is done with the document and requests the Close function, the program will merely go ahead and write the changes out to disk without stopping to ask for confirmation with the Save Changes dialog box.

In a perfect world, that would be enough, but computers and software can crash, power can fail, and other unpredictable, catastrophic events can conspire to erase your work. If the power fails before you have clicked Close, all your changes are lost as the memory containing them scrambles. The original copy on disk will be all right, but hours of work can still be lost. To keep this from happening, the program must also save the document at

© Copyright Virtual University of Pakistan

intervals during the user's session. Ideally, the program will save every single little change as soon as the user makes it, in other words, after each keystroke. For most programs, this is quite feasible. Most documents can be saved to hard disk in just a fraction of a second. Only for certain programs—word processors leap to mind — would this level of saving be difficult (but not impossible).

Word will automatically save according to a countdown clock, and you can set the delay to any number of minutes. If you ask for a save every *two* minutes, for example,

after precisely two minutes the program will stop to write your changes out to disk regardless of what you are doing at the time. If you are typing when the save begins, it just clamps shut in a very realistic and disconcerting imitation of a broken program. It is a very unpleasant experience. If the algorithm would pay attention to the user instead of the clock, the problem would disappear. Nobody types continuously. Everybody stops to gather his thoughts, or flip a page, or take a sip of coffee. All the program needs to do is wait until the user stops typing for a couple of seconds and *then* save.

This automatic saving every few minutes and at close time will be adequate for almost even body. Some people though, like the authors, are so paranoid about crashing and losing data that they habitually press Ctrl+S after *every* paragraph, and sometimes after every sentence (Ctrl+S is the keyboard accelerator for the manual save function). AM programs should have manual save controls, but users should not be *required* to invoke manual saves.

Right now, the save function is prominently placed on the primary program menu. The save dialog is forced on all users whose documents contain unsaved changes when users ask to close the document or to quit or exit the program. These artifacts must go away, but the manual save functionality can and should remain in place exactly as it is now.

Creating a copy of the document

This should be an explicit function called Snapshot Copy. The word snapshot makes it clear that the copy is identical to the original, while also making it clear that the copy is not tied to the original in any way. That is, subsequent changes to the original will have no effect on the copy. The new copy should automatically be given a name with a standard form like Copy of Alpha, where Alpha is the name of the original document. If there is already a document with that name, the new copy should be named Second Copy of Alpha. The copy should be placed in the same directory as the original.

It is very tempting to envision the dialog box that accompanies this command, but there should be no such interruption. The program should take its action quietly, efficiently, and sensibly, without badgering the user with silly questions like Make a Copy? In the user's mind it is a simple command. If there are any anomalies, the program should make a constructive decision on its own authority.

Naming and renaming the document

The name of the document should be shown on the application's title bar. If the user decides to rename the document, he can just click on it and edit it in place. What could be simpler and more direct than that?

Placing and moving the document

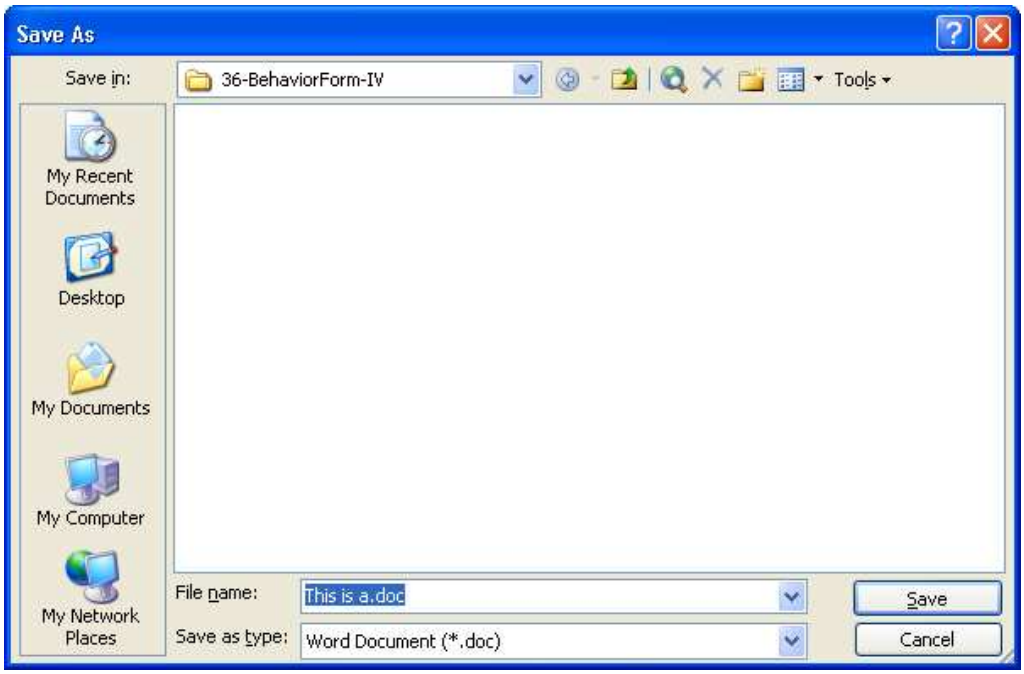
Most desktop productivity documents that are edited already exist. They are opened rather than created from scratch. This means that their position in the file system is already established. Although we think of establishing the home directory for a document at either the moment of creation or the moment of first saving, neither of these events is

© Copyright Virtual University of Pakistan

particularly meaningful outside of the implementation model. The new file should be put somewhere reasonable where the user can find it again. .
If the user wants to explicitly place the document somewhere in the file system hierarchy, he can request this function from the menu. A relative of the Save As dialog appears with the current document highlighted. The user can then move the file to any desired location. The program thus places all files automatically, and this dialog is used only to *move* them elsewhere.

Specifying the stored format of the document

There is an additional function implemented on the Save As dialog.



The combo box at the bottom of the dialog allows the user to specify the physical format of the file. This function should not be located here. By tying the physical format to the act of saving, the user is confronted with additional, unnecessary complexity added to saving. In Word, if the user innocently changes the format, both the save function and any subsequent close action is accompanied by a frightening and unexpected confirmation box. Overriding the physical format of a file is a relatively rare occurrence. Saving a file is a very common occurrence. These two functions should not be combined.
From the user's point-of-view, the physical format of the document—whether it is rich text, ASCII, or Word format, for example — is a characteristic of the document rather than of the disk file. Specifying the format shouldn't be associated with the act of saving the file to disk. It belongs more properly in a Document Properties dialog

The physical format of the document should be specified by way of a small dialog box callable from the main menu. This dialog box should have significant cautions built into its interface to make it clear to the user that the function could involve significant data loss.

© Copyright Virtual University of Pakistan

In the case of some drawing programs, where saving image files to multiple formats is desirable, an Export dialog (which some drawing programs already support) is appropriate for this function.

Reversing changes

If the user inadvertently makes changes to the document that must be reversed, the tool already exists for correcting these actions: undo. The file system should not be called in as a surrogate for undo. The file system may be the mechanism for supporting the function, but that doesn't mean it should be rendered to the user in those terms. The concept of going directly to the file system to undo changes merely undermines the undo function.
The milestoning function described later in this chapter shows how a file-centric vision of undo can be implemented so that it works well with the unified file model.

Abandoning all changes

It is not uncommon for the user to decide that she wants to discard all the changes she has made after opening or creating a document, so this action should be explicitly supported. Rather than forcing the user to understand the file system to achieve her goal, a simple Abandon Changes function on the main menu would suffice. Because this function involves significant data loss, the user should be protected with clear warning signs. Making this function undoable would also be relatively easy to implement and highly desirable.

37.2 Creating a milestone copy of the document

Milestoning is very similar to the Copy command. The difference is that this copy is managed by the application after it is made. The user can call up a Milestone dialog box that lists each milestone along with various statistics about it, like the time it was recorded and its length. With a click, the user can select a milestone and, by doing so, he also immediately selects it as the active document. The document that was current at the time of the new milestone selection will be milestoned itself, for example, under the name Milestone of Alpha 12/17/03, 1:53 PM. Milestoning is, in essence, a lightweight form of versioning.

A new File menu

New and Open function as before, but Close closes the document without a dialog box or any other fuss, after an automatic save of changes. Rename/Move brings up a dialog that lets the user rename the current file or move it to another directory. Make Snapshot Copy creates a new file that is a copy of the current document. Print collects all printer-related controls in a single dialog. Make Milestone is similar to Copy, except that the program manages these copies by way of a dialog box summoned by the Revert to Milestone menu item. Abandon Changes discards all changes made to the document since it was opened or created. Document Properties opens a dialog box that lets the user change the physical format of the document. Exit behaves as it does now, closing the document and application.

A new name for the File menu

Now that we are presenting a unified model of storage instead of the bifurcated implementation model of disk and RAM, we no longer need to call the left-most application menu *the File* menu — a reflection on the implementation model, not the user's model. There are two reasonable alternatives.

© Copyright Virtual University of Pakistan

We could label the menu after the type of documents the application processes. For example, a spreadsheet application might label its left-most menu Sheet. An invoicing program might label it Invoice,.
Alternatively, we can give the left-most menu a more generic label such as Document. This is a reasonable choice for applications like word processors, spreadsheets, and drawing programs, but may be less appropriate for more specialized niche applications.
Conversely, those few programs that do represent the contents of disks as files — generally operating system shells and utilities —*should* have a File menu because they are addressing files *as files*.

37.3 Are Disks and Files Systems a Feature?

From the user's point of view, there is no reason for disks to exist. From the hardware engineer's point of view, there are three:

- Disks are cheaper than solid-state memory.
- Once written to, disks don't forget when the power is off.
- Disks provide a physical means of moving information from one computer to another.

Reasons two and three are certainly useful, but they are also not the exclusive domains of disks. Other technologies work as well or better. There are varieties of RAM that don't forget their data when the power is turned off. Some types of solid-state memory can retain data with little or no power. Networks and phone lines can be used to physically transport data to other sites, often more easily than with removable disks.
Reason number one — cost — is the *real* reason why disks exist. Non-volatile solid-state memory is a lot more expensive than disk drives. Reliable, high-bandwidth networks haven't been around as long as removable disks, and they are more expensive.

Disk drives have many drawbacks compared to RAM. Disk drives are much slower than solid-state memory. They are much less reliable, too, because they depend on moving parts. They generally consume more power and take up more space, too. But the biggest problem with disks is that the computer, the actual CPU, can't directly read or write to them! Its helpers must first bring data into solid-state memory before the CPU can work with it. When the CPU is done, its helpers must once again move the data back out to the disk. This means that processing that involves disks is necessarily orders of magnitude slower and more complex than working in plain RAM.
The time and complexity penalty for using disks is so severe that nothing short of enormous cost-differential could compel us to rely on them. Disks do not make computers better, more powerful, faster, or easier to use. Instead, they make computers weaker, slower, and more complex. They are a compromise, a dilution of the solid-state architecture of digital computers. If computer designers could have economically used RAM instead of disks they would have done so without hesitation - and in fact they do, in the newest breeds of handheld communicators and PDAs that make use of Compact Plash and similar solid-state memory technologies.
Wherever disk technology has left its mark on the design of our software, it has done so for implementation purposes only, and not in the service of users or any goal-directed design rationale.

© Copyright Virtual University of Pakistan

37.4 Time for Change

There are only two arguments that can be mounted in favor of application software implemented in the file system model: Our software is already designed and built that way, and users are used to it
Neither of these arguments is valid. The first one is irrelevant because new programs written with a unified file model can freely coexist with the older implementation model applications. The underlying file system doesn't change at all. In much the same way that toolbars quickly invaded the interfaces of most applications in the last few years, the unified file model could also be implemented with similar success and user acclaim.
The second argument is more insidious, because its proponents place the user community in front of them like a shield. What's more, if you ask users themselves, they will reject the new solution because they abhor change, particularly when that change affects something they have already worked hard to master — like the file system. However, users are not always the best predictors of design successes, especially when the designs are different from anything they've already experienced,
In the eighties, Chrysler showed consumers early sketches of a dramatic new automobile design: the minivan. The public gave a uniform thumbs-down to the new design. Chrysler went ahead and produced the Caravan anyway, convinced that the design was superior. They were right, and the same people who initially rejected the
design have not only made the Caravan the one of the best-selling minivans, but also made the minivan the most popular new automotive archetype since the convertible.
Users aren't interaction designers, and they cannot be expected to visualize the larger effects of interaction paradigm shifts. But the market has shown that people will gladly give up painful, poorly designed software for easier, better software even if they don't understand the explanations behind the design rationale

37.5 Making Software Considerate

Two Stanford sociologists, Clifford Nass and Byron Reeves, discovered that humans seem to have instincts that tell them how to behave around other sentient beings. As soon as any artifact exhibits sufficient levels of interactivity — such as that found in your average software application — these instincts are activated. Our reaction to software as sentient is both unconscious and unavoidable.
The implication of this research is profound: If we want users to like our software, we should design it to behave in the same manner as a likeable person. If we want users to be productive with our software, we should design it to behave like a supportive human colleague.

Designing Considerate Software

Nass and Reeves suggest that software should be polite, but the term considerate is preferred. Although politeness could be construed as a matter of protocol —saying please and thank you, but doing little else helpful — being truly considerate means putting the needs of others first. Considerate software has the goals and needs of its users as its primary concern beyond its basic functions.
If software is stingy with information, obscures its process, forces the user to hunt around for common functions, and is quick to blame the user for its own failings, the user will dislike the software and have an unpleasant experience. This will happen regardless of how cute, how representational, how visually metaphoric, how content-filled, or how anthropomorphic the software is.
On the other hand, if the interaction is respectful, generous, and helpful, the user will like the software and will have a pleasant experience. Again, this will happen regardless of the composition of the interface; a green-screen command-line interface will be well liked if it can deliver on these other points.

© Copyright Virtual University of Pakistan

What Makes Software Considerate?

Humans have many wonderful characteristics that make them considerate but whose definitions are fuzzy and imprecise. The following list enumerates some of the characteristics of considerate interactions that software-based products (and humans) should possess:

- Considerate software takes an interest.
- Considerate software is deferential.
- Considerate software is forthcoming.
- Considerate software uses common sense.
- Considerate software anticipates needs.
- Considerate software is conscientious.
- Considerate software doesn't burden you with its personal problems.
- Considerate software keeps you informed.
- Considerate software is perceptive.
- Considerate software is self-confident.
- Considerate software doesn't ask a lot of questions.
- Considerate software takes responsibility.
- Considerate software knows when to bend the rules.

Well now discuss the characteristics in detail.

Considerate software takes an interest

A considerate friend wants to know more about you. He remembers likes and dislikes so he can please you in the future. Everyone appreciates being treated according to his or her own personal tastes

Most software, on the other hand, doesn't know or care who is using it. Little, if any, of the *personal* software on *our personal* computers seems to remember anything about us, in spite of the fact that we use it constantly, repetitively, and exclusively.

Software should work hard to remember our work habits and, particularly, everything that we say to it. To the programmer writing the program, it's a just-in-time information world, so when the program needs some tidbit of information, it simply demands that the user provide it. The-program then discards that tidbit, assuming that it can merely ask for it again if necessary. Not only is the program better suited to remembering than the human, the program is also *inconsiderate* when, acting as a supposedly helpful tool, it forgets.

Considerate software is deferential

A good service provider defers to her client. She understands the person she is serving is the boss. When a restaurant host shows us to a table in a restaurant, we consider his choice of table to be a suggestion, not an order. If we politely request another table in an otherwise empty restaurant, we expect to be accommodated. If the host refuses, we are likely to choose a different restaurant where our desires take precedence over the host's.

Inconsiderate software supervises and passes judgment on human actions. Software is within its rights to express its *opinion* that we are making a mistake, but it is being presumptuous when it judges our actions. Software can *suggest* that we not Submit our entry until we've typed in our telephone number. It should also explain the consequences, but if we wish to Submit without the number, we expect the software to do as it is told. (The very word *Submit* and the concept it stands for are a reversal of the deferential role.

© Copyright Virtual University of Pakistan

The software should submit to the user, and any program that proffers a Submit button is being rude. Take notice, almost every transactional site on the World Wide Web!)

Considerate software is forthcoming

If we ask a store employee where to locate an item, we expect him to not only answer the question, but to volunteer the extremely useful collateral information that a more expensive, higher quality item like it is currently on sale for a similar price.

Most software doesn't attempt to provide related information. Instead, it only narrowly answers the precise questions we ask it, and it is typically not forthcoming about other information even if it is clearly related to our goals. When we tell our word processor to print a document, it doesn't tell us when the paper supply is low, or when forty other documents are queued up before us, or when another nearby printer is free. A helpful human would.

Considerate software uses common sense

Offering inappropriate functions in inappropriate places is a hallmark of software-based products. Most software-based products put controls for constantly used functions adjacent to never-used controls. You can easily find menus offering simple, harmless functions adjacent to irreversible ejector-seat-lever expert functions. It's like seating you at a dining table right next to an open grill.

Considerate software anticipates needs

A human assistant knows that you will require a hotel room when you travel to another city, even when you don't ask explicitly. She knows the kind of room you like and reserves one without any request on your part. She anticipates needs.

A Web browser spends most of its time idling while we peruse Web pages. It could easily anticipate needs and prepare for them while we are reading. It could use that idle time to preload all the links that are visible. Chances are good that we will soon ask the browser to examine one or more of those links. It is easy to abort an unwanted request, but it is always time-consuming to wait for a request to be filled.

Considerate software is conscientious

A conscientious person has a larger perspective on what it means to perform a task. Instead of just washing the dishes, for example, a conscientious person also wipes down the counters and empties the trash because those tasks are also related to the larger *goal:* cleaning up the kitchen. A conscientious person, when drafting a report, also puts a handsome cover page on it and makes enough photocopies for the entire department.

Considerate software doesn't burden you with its personal problems

At a service desk, the agent is expected to keep mum about her problems and to show a reasonable interest in yours. It might not be fair to be so one-sided, but that's the nature of the service business. Software, too, should keep quiet about its problems and show interest in ours. Because computers don't have egos or tender sensibilities, they should be perfect in this role; but they typically behave the opposite way.

© Copyright Virtual University of Pakistan

Software whines at us with error messages, interrupts us with confirmation dialog boxes, and brags to us with unnecessary notifiers (Document Successfully Saved! How nice for you, Mr. Software: Do you ever *unsuccessfully* save?). We aren't interested in the program's crisis of confidence about whether or not to purge its Recycle bin. We don't want to hear its whining about not being sure where to put a file on disk. We don't need to see information about the computer's data transfer rates and its loading sequence, any more than we need information about the customer service agent's unhappy love affair. Not only should software keep quiet about its problems, but it should also have the intelligence, confidence, and authority to fix its problems on its own.

Considerate software keeps us informed

Although we don't want our software pestering us incessantly with its little fears and triumphs, we do want to be kept informed about the things that matter to *us*. Software can provide us with modeless feedback about what is going on.

Considerate software is perceptive

Most of our existing software is not very perceptive. It has a very narrow understanding of the scope of most problems. It may willingly perform difficult work, but only when given the precise command at precisely the correct time. If, for example, you ask the inventory query system to tell you how many widgets are in stock, it will dutifully ask the database and report the number as of the time you ask. But what if, twenty minutes later, someone in your office cleans out the entire stock of widgets. You are now operating under a potentially embarrassing misconception, while your computer sits there, idling away billions of wasted instructions. It is not being perceptive. If you want to know about widgets once, Isn't that a good clue that you probably will want to know about widgets again? You may not want to hear widget status reports every day for the rest of your life, but maybe you'll want to get them for the rest of the week. Perceptive software observes what the user is doing and uses those patterns to offer relevant information.

Software should also watch our preferences and remember them without being asked explicitly to do so. If we always maximize an application to use the entire available screen, the application should get the idea after a few sessions and always launch in that configuration. The same goes for placement of palettes, default tools, frequently used templates, and other useful settings.

Considerate software is self-confident

Software should stand by its convictions. If we tell the computer to discard a file, It shouldn't ask, "Are you sure?" Of course we're sure, otherwise we wouldn't have asked. It shouldn't second-guess itself or us.

On the other hand, if the computer has any suspicion that we might be wrong (which Is always), it should anticipate our changing our minds by being prepared to undelete the file upon our request.

How often have you clicked the Print button and then gone to get a cup of coffee, only to return to find a fearful dialog box quivering in the middle of the screen asking, "Are you sure you want to print?" This insecurity is infuriating and the antithesis of considerate human behavior.

Considerate software doesn't ask a lot of questions

Inconsiderate software asks lots of annoying questions- Excessive choices quickly stop being a benefit and become an ordeal.

© Copyright Virtual University of Pakistan

Choices can be offered in different ways. They can be offered in the way that we window shop. We peer in the window at our leisure, considering, choosing, or ignoring the goods offered to us — no questions asked. Alternatively, choices can be forced on us like an interrogation by a customs officer at a border crossing: *"Do you have anything to declare?"* We don't know the consequences of the question. Will we be searched or not? Software should never put users through this kind of intimidation.

Considerate software fails gracefully

When a friend of yours makes a serious faux pas, he tries to make amends later and undo what damage can be undone. When a program discovers a fatal problem, it has the choice of taking the time and effort to prepare for its failure without hurting the user, or it can simply crash and burn. In other words, it can either go out like a psychotic postal employee, taking the work of a dozen coworkers and supervisors with it, or it can tidy up its affairs, ensuring that as much data as possible is preserved in a recoverable format.

Most programs are filled with data and settings. When they crash, that information is normally just discarded. The user is left holding the bag. For example, say a program is computing merrily along, downloading your e-mail from a server when it runs out of memory at some procedure buried deep in the internals of the program. The program, like most desktop software, issues a message that says, in effect, "You are completely hosed," and terminates immediately after you click OK. You restart the program, or sometimes the whole computer, only to find that the program lost your e-mail and, when you interrogate the server, you find that it has also erased your mail because the mail was already handed over to your program. This is not what we should expect of good software.

In our e-mail example, the program accepted e-mail from the server — which then erased its copy — but didn't ensure that the e-mail was properly recorded locally. If the e-mail program had made sure that those messages were promptly written to the local disk, even before it informed the server that the messages were successfully downloaded, the problem would never have arisen.

Even when programs don't crash, inconsiderate behavior is rife, particularly on the Web. Users often need to enter detailed information into a set of forms on a page. After filling in ten or eleven fields, a user might press the Submit button, and, due to some mistake or omission on his part, the site rejects his input and tells him to correct it. The user then clicks the back arrow to return to the page, and lo, the ten valid entries were inconsiderately discarded along with the single invalid one.

Considerate software knows when to bend the rules

When manual information processing systems are translated into computerized systems, something is lost in the process. Although an automated order entry system can handle millions more orders than a human clerk can, the human clerk has the ability to *work the system* in a way most automated systems ignore. There is almost never a way to jigger the functioning to give or take slight advantages in an automated system.

In a manual system, when the clerk's friend from the sales force calls on the phone and explains that getting the order processed speedily means additional business, the clerk can expedite that one order. When another order comes in with some critical information missing, the clerk can go ahead and process it, remembering to acquire and record the information later. This flexibility is usually absent from automated systems.

In most computerized systems, there are only two states: non existence or full-compliance. No intermediate states are recognized or accepted. In any manual system, there

© Copyright Virtual University of Pakistan

is an important but paradoxical state — unspoken, undocumented, but widely relied upon — of suspense, wherein a transaction can be accepted although still not being fully processed. The human operator creates that state in his head or on his desk or in his back pocket.

For example, a digital system needs both customer and order information before it can post an invoice. Whereas the human clerk can go ahead and post an order in advance of detailed customer information, the computerized system will reject the transaction, unwilling to allow the invoice to be entered without it.

The characteristic of manual systems that let humans perform actions out of sequence or before prerequisites are satisfied is called fudge ability. It is one of the first casualties when systems are computerized, and its absence is a key contributor to the inhumanity of digital systems. It is a natural result of the implementation model. The programmers don't see any reason to create intermediate states because the computer has no need for them. Yet there are strong human needs to be able to bend the system slightly.

One of the benefits of fudgeable systems is the reduction of mistakes. By allowing many small temporary mistakes into the system and entrusting humans to correct them before they cause problems downstream, we can avoid much bigger, more permanent mistakes. Paradoxically, most of the hard-edged rules enforced by computer systems are imposed to prevent just such mistakes. These inflexible rules cast the human and the software as adversaries, and because the human is prevented from fudging to prevent big mistakes, he soon stops caring about protecting the software from really colossal problems. When inflexible rules are imposed on flexible humans, both sides lose. It is invariably bad for business to prevent humans from doing what they want, and the computer system usually ends up having to digest invalid data anyway.

In the real world, both missing information and extra information that doesn't fit into a standard field are important tools for success. Information processing systems rarely handle this real-world data. They only model the rigid, repeatable data portion of transactions, a sort of skeleton of the actual transaction, which may involve dozens of

meetings, travel and entertainment, names of spouses and kids, golf games and favorite sports figures. Maybe a transaction can only be completed if the termination date is extended two weeks beyond the official limit. Most companies would rather fudge on the termination date than see a million-dollar deal go up in smoke. In the real world, limits are fudged all the time. Considerate software needs to realize and embrace this fact.

Considerate software 'takes' responsibility

Too much software takes the attitude: "It isn't my responsibility." When it passes a job along some hardware device, it washes its hands of the action, leaving the stupid hardware to finish up. Any user can see that the software isn't being considerate or conscientious, that the software isn’t shouldering its part of the burden for helping the user become more effective.

In a typical print operation, for example, a program begins sending the 20 pages of a report to the printer and simultaneously puts up a print process dialog box with a Cancel button. If the user quickly realizes that he forgot to make an important change, he clicks the Cancel button just as the first page emerges from the printer. The program immediately cancels the print operation. But unbeknownst to the user, while the printer was beginning to work on page 1, the compute has already sent 15 pages into the printer's buffer. The program cancels the last five pages, but the printer doesn't know anything about the cancellation; it just knows that it was sent 15 pages, so it goes ahead and prints them. Meanwhile, the program smugly tells the user that the function was canceled. The program lies, as the user can plainly see.

The user isn't very sympathetic to the communication problems between the applications am the printer. He doesn't care that the communications are one-way. All he knows is that he decides

© Copyright Virtual University of Pakistan

not to print the document before the first page appeared in the printer's output basket, he clicked the Cancel button, and then the stupid program continued printing for 15 pages even though hi acted in plenty of time to stop it. It even acknowledged his Cancel command. As he throws the 15 wasted sheets of paper in the trash, he growls at the stupid program.

Imagine what his experience would be if the application could communicate with the print driver and the print driver could communicate with the printer. If the software were smart enough the print job could easily have been abandoned before the second sheet of paper was wasted. The printer certainly has a Cancel function — it's just that the software is too indolent to use it, because its programmers were too indolent to make the connection.

37.6 Considerate Software Is possible

Our software-based products irritate us because they aren't considerate, not because they lack features. As this list of characteristics shows, considerate software is usually no harder to build than rude or inconsiderate software. It simply means that someone has to envision interaction that emulates the qualities of a sensitive and caring friend. None of these characteristics is at odds with the other, more obviously pragmatic goals of business computing. Behaving more humanely can be the most pragmatic goal of all.

37.7 Making Software Smarts:

Because every instruction in every program must pass single-file through the CPU, we tend to optimize our code for this needle's eye. Programmers work hard to keep the number of instructions to a minimum, assuring snappy performance for the user. What we often forget, however, is that as soon as the CPU has hurriedly finished all its work, it waits idle, doing nothing, until the user issues another command. We invest enormous efforts in reducing the computer's reaction time, but we invest little or no effort in putting it to work proactively when it is not busy reacting to the user. Our software commands the CPU as though it were in the army, alternately telling it to hurry up and wait. The hurry up part is great, but the waiting needs to stop.

The division of labor in the computer age is very clear: The computer does the work, and the user does the thinking. Computer scientists tantalize us with visions of artificial intelligence: computers that think for themselves. However, users don't really need much help in the thinking department. They *do* need a lot of help with the work of information management—activities like finding and organizing information — but the actual decisions made from that information are best made by the wetware — us.

There is some confusion about smart software. Some naive observers think that smart software is actually capable of behaving intelligently, but what the term really means is that these programs are capable of working hard even when conditions are difficult and even when the user isn't busy. Regardless of our dreams of thinking computers, there is a much greater and more immediate opportunity in simply getting our computers to work harder. This lecture discusses some of the most important ways that software can work a bit harder to serve humans better.

37.8 Putting the Idle Cycles to Work

In our current computing systems, users need to remember too many things, such as the names they give to files and the precise location of those files in the file system. If a user wants to find that spreadsheet with the quarterly projections on it again, he must either remember its name or go browsing. Meanwhile, the processor just sits there, wasting billions of cycles.

Most current software also takes no notice of context. When a user is struggling with a particularly difficult spreadsheet on a tight deadline, for example, the program offers precisely as much help

© Copyright Virtual University of Pakistan

as it offers when he is noodling with numbers in his spare time. Software can no longer, in good conscience, waste so much idle time while the user works. It is time for our computers to begin to shoulder more of the burden of work in our day-to-day activities.

Wasted cycles

Most users in normal situations can't do anything in less than a few seconds. That is enough time for a typical desktop computer to execute at least a *billion* instructions. Almost without fail, those interim cycles are dedicated to idling. The processor does *nothing* except wait. The argument against putting those cycles to work has always been: "We can't make assumptions; those assumptions might be wrong." Our computers today are so powerful that, although the argument is stilt true, it is frequently irrelevant. Simply put, it doesn't matter if the program's assumptions are wrong; it has enough spare power to make several assumptions and discard the results of the had ones when the user finally makes his choice.

With Windows and Mac OS X's pre-emptive, threaded multitasking, you can perform extra work in the background without affecting the performance the user sees. The program can launch a search for a file, and if the user begins typing, merely abandon it until the next hiatus. Eventually, the user stops to think, and the program will have time to scan the whole disk. The user won't even notice.

Every time the program puts up a modal dialog box, it goes into an idle waiting state, doing no work while the user struggles with the dialog. This should never happen. It would not be hard for the dialog box to hunt around and find ways to help. What did the user do last time? The program could, for example, offer the previous choice as a suggestion for this time.

We need a new, more proactive way of thinking about how software can help people with their goals and tasks.

Putting the cycles to better use

If your program, Web site, or device could predict what the user is going to do next, couldn't it provide a better interaction? If your program could know which selections the user will make in a particular dialog box or form, couldn't that part of the interface be skipped? Wouldn't you consider advance knowledge of what actions your users take to be an awesome secret weapon of interface design?

Well, you can predict what your users will do. You *can* build a sixth sense into your program that will tell it with uncanny accuracy exactly what the user will do next! All those billions of wasted processor cycles can be put to great use: All you need to do is give your interface a memory.

37.9 Giving Software a Memory

When we use the term memory in this context, we don't mean RAM, but rather a program facility for tracking and responding to user actions over multiple sessions. If your program simply remembers what the user did the last time (and how), it can use that remembered behavior as a guide to how it should behave the next time. As we'll see later in this chapter, your program should remember more than one previous action, but this simple principle is one of the most effective tools available for designing software behavior.

You might think that bothering with a memory isn't necessary; it's easier to just ask the user each time. Programmers are quick to pop up a dialog box to request any information that isn't lying conveniently around. But as we discussed, *people don't like to be asked questions.*

© Copyright Virtual University of Pakistan

Continually interrogating users is not only a form of excise, but from a psychological perspective, it is a subtle way of expressing doubt about their authority.

Most software is forgetful, remembering little or nothing from execution to execution. If our programs *are* smart enough to retain any information during and between uses, it is usually information that makes the job easier for *the programmer and* not for the user. The program willingly discards information about the way it was used, how it was changed, where it was used, what data g it processed, who used it, and whether and how frequently the various facilities of the program were used. Meanwhile, the program fills initialization files with driver-names, port assignments, and other details that ease the programmer's burden. It is possible to use the exact same facilities to dramatically increase the smarts of your software from the perspective of the user.

37.10 Task Coherence

Does this kind of memory really work? Predicting what a user will do by remembering what he did -last is based on the principle of task coherence: the idea that our goals and the way we achieve them (via tasks) is generally the same from day to day. This is not only true for tasks like brushing our teeth and eating our breakfasts, but it also describes how we use our word processors, e-mail programs, cell phones, and e-commerce sites.

When a consumer uses your product, there is a high-percentage chance that the functions he uses and the way he uses them will be very similar to what he did last time he used your program. He may even be working on the same documents, oral least the same types of documents, located similar places. Sure, he won't be doing the exact same thing each time, but it will likely be variant of a small number of repeated patterns. With significant reliability, you can predict the behavior of your users by the simple expedient of remembering what they did the last several times they used the program. This allows you to greatly reduce the number of questions your program must M

Remembering choices and defaults

The way to determine what information the program should remember is with a simple rule: If it's worth the user entering, it's worth the program remembering.

Any time your program finds itself with a choice, and especially when that choice is being offered to the user, the program should remember the information from run to run. Instead of choosing a hard-wired default, the program can use the previous setting as the default, and it will have a much better chance of giving the user what he wanted. Instead of asking the user to make a determination, the program should go ahead and make the same determination the user made last time, and let the user change it if it was wrong. Whatever options the user set should be remembered, so that the options remain in effect until manually changed. If the user ignored facilities of the program or turned them off, they should not be offered to the user again. The user will seek them out when and if he is ready for them.

One of the most annoying characteristics of programs without memories is that they are so parsimonious with their assistance regarding files and disks. If there is one place where the user needs help, it's with files and disks. A program like Word remembers the last place the user looked for a file. Unfortunately, if the user always puts his files in a directory called Letters, then edits a document template stored in the Template directory just one time, all his subsequent letters will be stored in the Template directory rather than in the Letters directory. So the program must remember more than just the last place the files were accessed. It must remember the last place files *of each type* were accessed.

© Copyright Virtual University of Pakistan

The position of windows should also be remembered, so if you maximized the document last time it should be maximized next time. If the user positioned it next to another window, it is positioned the same way the next time without any instruction from the user. Microsoft Office applications now do a good job of this.

Remembering patterns

The user can benefit in several ways from a program with a good memory. Memory reduces excise, the useless effort that must be devoted to managing the tool and not doing the work. A significant portion of the total excise of an interface is in having to explain things to the program that it should already know. For example, in your word processor, you might often reverse-out text, making it white on black. To do this, you select some text and change the font color to white. Without altering the selection, you then set the background color to black. If the program paid enough attention, it would notice the fact that you requested two formatting steps without an intervening selection option. As far as you're concerned, this is effectively a single operation. Wouldn't it be nice if the program, upon seeing this unique pattern repeated several times, automatically created a new format style of this type — or better yet, created a new Reverse-Out toolbar control?

Most mainstream programs allow their users to set defaults, but this doesn't fit the hill as a memory would. Configuration of this kind is an onerous process for all but power users, and many users will never understand how to customize defaults to their liking

37.11 Actions to Remember

Everything that users do should be remembered. There is plenty of storage on our hard drives/ and a memory for your program is a good investment of storage space. We tend to think that programs are wasteful of disk space because a big horizontal application might consume 30 or 40 MB of space. That is typical usage for a program, but not for user data. If your word processor saved 1 KB of execution notes every time you ran it, it still wouldn't amount to much. Let's say that you use your word processor ten times every business day. There are approximately 200 workdays per year, so you run the program 2,000 times a year. The net consumption is still only 2 MB, and that gives an exhaustive recounting of the entire year! This is probably not much more than the background image you put on your desktop.

File locations

All file-open facilities should remember where the user gets his files. Most users only access files from a few directories for each given program. The program should remember these source directories and offer them on a combo box on the

File-Open dialog. The user should never have to step through the tree to a given directory more than once.

Deduced information

Software should not simply remember these kinds of explicit facts, but should also remember useful information that can be deduced from these facts. For example, if the program remembers the number of bytes changed in the file each time it is opened, it can help the user with some reasonableness checks. Imagine that the changed-byte-count for a file was 126, 94, 43, 74, 81, 70, 110, and 92. If the user calls up the file and changes 100 bytes, nothing would be out of the ordinary." But if the number of changed bytes suddenly shoots up to 5000, the program might

© Copyright Virtual University of Pakistan

suspect that something is amiss. Although there is a chance that the user has inadvertently done something about which he will be sorry, the probability of that is low, so it isn't right to bother him with a confirmation dialog. It is, however, very reasonable for the program to make sure to keep a milestone copy of the file before the 5000 bytes were changed, just in case. The program probably won't need to keep it beyond the next time the user accesses that file, because the user will likely spot any mistake that glaring immediately, and he would then demand an undo.

Multi-session undo

Most programs discard their stack of undo actions when the user closes the document or the program. This is very shortsighted on the program's part. Instead, the program could write the undo stack to a file. When the user reopens the file, the program could reload its undo stack with the actions the user performed the last time the program was run — even if that was a week ago!

Past data entries

A program with a better memory can reduce the number of errors the user makes. This is simply because the user has to enter less information. More of it will be entered automatically from the program's memory- In an invoicing program, for example, if the software enters the date, department number, and other standard fields from memory, the user has fewer opportunities to make typing errors in these fields.

If the program remembers what the user enters and uses that information for future reasonableness checks, the program can work to keep erroneous data from being entered. Imagine a data entry program where zip codes and city names are remembered from run to run. When the user enters a familiar city name along with an unfamiliar zip code, the field can turn yellow, indicating uncertainty about the match. And when the user enters a familiar city name with a zip code already associated with another city, the field can turn pink, indicating a more serious ambiguity. He wouldn't necessarily have to take any action because of these colors, but the warning is there if he wants it.

Some Windows 2000 and XP applications, notably Internet Explorer, have a facility of similar nature: Named data entry fields remember what has been entered into them before, and allow the user to pick those values from a combo box. For security-minded individuals, this feature can be turned off, but for the rest of us, it saves time and prevents errors.

Foreign application activities on program files

Applications might also leave a small thread running between invocations. This little program can keep an eye on the files it worked on. It can track where they go and who reads and writes to them. This information might be helpful to the user when he next runs the application. When he tries to open a particular file, the program can help him find it, even if it has been moved. The program can keep the user informed about what other functions were performed on his file, such as whether or not it was printed or faxed to someone. Sure, this information might not be needed, but the computer can easily spare the time, and it's only bits that have to be thrown away, after all.

37.12 Applying Memory to Your Applications

A remarkable thing happens to the software design process when developers accept the power of task coherence. Designers find that their thinking takes on a whole new quality.

© Copyright Virtual University of Pakistan

The normally unquestioned recourse of popping up a dialog box gets replaced with a more studied process, where the designer asks questions of much greater subtlety. Questions like: How *much* should the program remember? Which aspects should be remembered? Should the program remember more than just the last setting? What constitutes a change in pattern? Designers start to imagine situations like this: The user accepts the same date format *50* times in a row, and then manually enters a different format once. The next time the user enters a date, which format should the program use? The format used 50 times or the more recent one-time format? How many times must the new format be specified before it becomes the default? Just because there is ambiguity here, the program still shouldn't ask the user. It must use its initiative to make a reasonable decision. The user is free to override the program's decision if it is the wrong one.

The following sections explain some characteristic patterns in the ways people make choices that can help us resolve these more complex questions about task coherence.

Decision-set reduction

People tend to reduce an infinite set of choices down to a small, finite set of choices. Even when you don't do the exact same thing each time, you will tend to choose your actions from a small, repetitive set of options. People unconsciously perform this decision-set reduction, but software can take notice and act upon it.

For example, just because you went shopping at Safeway yesterday doesn't necessarily mean that you will be shopping at Safeway exclusively. However, the next time you need groceries, you will probably shop at Safeway again. Similarly, even though your favorite Chinese restaurant has 250 items on the menu, chances are that you will usually choose from your own personal subset of five or six favorites. When people drive to and from work, they usually choose from a small number of favorite routes, depending on traffic conditions. Computers, of course, can remember four or five things without breaking a sweat.

Although simply remembering the last action is better than not remembering anything, it can lead to a peculiar pathology if the decision-set consists of precisely two elements. If, for example, you alternately read files from one directory and store them in another, each time the program offers you the last directory, it will be guaranteed to be wrong. The solution is to remember more than just one previous choice.

Decision-set reduction guides us to the idea that pieces of information the program must remember about the user's choices tend to come in groups. Instead of there being one right way, there will be several options that are all correct. The program should look for more subtle clues to differentiate which one of the small set is correct. For example, if you use a check-writing program to pay your bills, the program may very quickly learn that only two or three accounts are used regularly. Rut how can it determine from a given check which of the three accounts is the most likely to be appropriate? If the program remembers the payees and amounts on an account-by-account basis, that decision would be easy. Every time you pay the rent, it is the exact same amount! It's the same with a car payment. The amount paid to the electric company might vary from check to check, but it probably stays within 10 or 20 percent of the last check written to them. All this information can be used to help the program recognize what is going on, and use that information to help the user.

Preference thresholds

The decisions people make tend to fall into two primary categories: important and unimportant. Any given activity may involve potentially hundreds of decisions, but only a

© Copyright Virtual University of Pakistan

very few of them are important. All the rest are insignificant. Software interfaces can use this idea of preference thresholds to simplify tasks for users.

After you decide to buy that car, you don't really care who finances it as long as the terms are competitive. After you decide to buy groceries, the particular checkout aisle you select is not important. After you decide to ride the Matterhorn, you don't really care which toboggan they seat you in.

Preference thresholds guide us in our user interface design by demonstrating that asking the user for successively detailed decisions about a procedure is unnecessary. After the user asks to print, we don't have to ask him how many copies he wants or whether the image is landscape or portrait. We can make an assumption about these things the first time out, and then remember them for all subsequent invocations. If the user wants to change them, he can always request the Printer Options dialog box.

Using preference thresholds, we can easily track which facilities of the program the user likes to adjust and which are set once and ignored. With this knowledge, the program can offer choices where it has an expectation that the user will want to take control, not bothering the user with decisions he won't care about.

Mostly right, most of the time

Task coherence predicts what the user will do in the future with reasonable, but not absolute, certainty. If our program relies on this principle, its natural to wonder about the uncertainty of our predictions. If we can reliably predict what the user will do 80% of the time, it means that 20% of the time we will be wrong. It might seem that the proper step to take here is to offer the user a choice, but this means that the user will be bothered by an unnecessary dialog 80% of the time, Rather than offering a choice, the program should go ahead and do what it thinks is most appropriate and allow the user to

override or undo it. If the undo facility is sufficiently easy to use and understand, the user won't be bothered by it. After all, he will have to use undo only two times out of ten instead of having to deal with a redundant dialog box eight times out of ten. This is a much better deal for humans.

37.13 Memory Makes a Difference

One of the main reasons our software is often so difficult to use is because its designers have made rational, logical assumptions that, unfortunately, are very wrong. They assume that the behavior of users is random and unpredictable, and that users must be interrogated to determine the proper course of action. Although human behavior certainly isn't deterministic like that of a digital computer, it is rarely random, and asking silly questions is predictably frustrating for users.

However, when we apply memory via task coherence to our software, we can realize great advantages in user efficiency and satisfaction. We would all like to have an assistant who is intelligent and self-motivated, one who shows initiative and drive, and who demonstrates good judgment and a keen memory. A program that makes effective use of its memory would be more like that self-motivated assistant, remembering helpful information and personal preferences from execution to execution without needing to ask. Simple things can make a big difference: the difference between a product your users tolerate, and one that they *love*. The next time you find your program asking your users a question, make it ask itself one instead.

© Copyright Virtual University of Pakistan

Lecture 38.

Behavior & Form – Part VI

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the Principles of visual interface design

38.1 Designing Look and Feel

The commonly accepted wisdom of the post-Macintosh era is that graphical user interfaces, or GUIs, are better than character-based user interfaces. However, although there are certainly GUI programs that dazzle us with their ease of use and their look and feel, most GUI programs still irritate and annoy us in spite of their graphical nature. It's easy enough, so it seems, to create a program with a graphical user interface that has a difficulty-of-use on par with a command-line Unix application. Why is this the case?
To find an answer to this question, we need to better understand the role of visual design in the creation of user interfaces.

Visual Art versus Visual Design

Practitioners of visual art and practitioners of visual design share a visual medium. Each must be skilled and knowledgeable about that medium, but there the similarity ends. The goal of the artist is to produce an observable artifact that provokes an aesthetic response. Art is thus a means of self-expression on topics of emotional or intellectual concern to the artist, and sometimes, to society at large. Few constraints are imposed on the artist; and the more singular and unique the product of the artist's exertions, the more highly it is valued. Designers, on the other hand, create artifacts that meet the goals of people other than themselves. Whereas the concern of contemporary artists is primarily *expression* of ideas or emotions, visual designers, as Kevin Mullet and Darrell Sano note in their excellent *book Designing Visual Interfaces* (1995), "are concerned with finding the *representation* best suited to the communication of some specific information." Visual interface designers, moreover, are concerned with finding the representation best suited to communicating the *behavior* of the software that they are designing.
-.

Graphic Design and Visual Interface Design

Design of user interfaces does not entirely exclude aesthetic concerns, but rather it places such' concerns within the constraints of a functional framework. Visual design in an interface context thus requires several related skills, depending on the scope of the interface in question. Any designer working on interfaces needs to understand the basics: color, typography, form, and composition. However, designers working on interfaces also need some understanding of interaction the behavior of the software, as well. It is rare to find visual designers with an even balance of these skills, although both types of visual perspectives are required for a truly successful interactive design

© Copyright Virtual University of Pakistan

Graphic design and user interfaces

Graphic design is a discipline that has, until the last twenty years or so, been dominated by the medium of print, as applied to packaging, advertising, and document design. Old-school graphic designers are uncomfortable designing in a digital medium and are unused to dealing with graphics at the pixel level, a requirement for most interface-design issues. However, a new breed of graphic designers has been trained in digital media and quite successfully applies the concepts of graphic design to the new, pixilated medium.

Graphic designers typically have a strong understanding of visual principles and a weaker understanding of concepts surrounding software behavior and interaction over time. Talented, digitally-fluent graphic designers excel at providing the sort of rich, clean, visually consistent, aesthetically pleasing, and exciting interfaces we see in Windows XP, Mac OS X, and some of the more visually sophisticated computer-game interfaces and consumer-oriented applications. These designers excel at creating beautiful and appropriate *surfaces* of the interface and are also responsible for the interweaving of corporate branding into software look and feel. For them, design is first about legibility and readability of information, then about tone, style, and framework that communicate a brand, and finally about communicating behavior through affordances.

Visual interface design and visual information design

Visual interface designers share some of the skills of graphic designers, but they focus more on the organizational aspects of the design and the way in which affordances communicate behavior to users. Although graphic designers are more adept at defining the *syntax* of the visual design— what it looks like — visual interface designers are more knowledgeable about principles of interaction. Typically, they focus on how to match the visual structure of the interface to the logical structure of both the user's and the program's behavior. Visual interface designers are also concerned with communication of program states to the user and with cognitive issues surrounding user perception of functions (layout, grids, figure-ground issues, and so on).

Visual *information* designers fulfill a similar role regarding content and navigation rather than more interactive functions. Their role is particularly important in Web design, where content often outweighs function. Their primary focus tends to be on controlling information hierarchy through the use of visual language. Visual information designers work closely with information architects, just as visual interface designers work closely with interaction designers,

Industrial design

Although it is beyond the scope of this book to discuss industrial design issues in any depth, as interactive appliances and handheld devices become widespread, industrial design is playing an ever-growing role in the creation of new interactive products. Much like the difference in skills between graphic designers and visual interface and information designers, there is a similar split among the ranks of industrial designers. Some are more adept at the creation of arresting and appropriate shapes and skins of objects, whereas others' talents lie more in the logical and ergonomic mapping of physical controls in a manner that matches user behaviors and communicates device behaviors. As more physical artifacts become software-enabled and sport sophisticated visual displays, it will become more important that interaction designers, industrial designers, and visual designers of all flavors work closely together to produce usable products.

© Copyright Virtual University of Pakistan

38.2 Principles of Visual Interface Design

The human brain is a superb pattern-processing computer, making sense of the dense quantities of visual information that bombard us everywhere we look. Our brains manage this chaotic input by discerning visual patterns and establishing a system of priorities for the things we see which in turn allows us to make conscious sense of the visual world. The ability of the

brain's visual system to assemble portions of our visual field into patterns based on visual cues is what allows us to process visual information so quickly and efficiently. Visual interface design must take advantage of our innate visual processing capabilities to help programs communicate their behavior and function to users.

There are some important principles that can help make your visual interface as easy and pleasurable to use as possible. Kevin Mullet and Darrell Sano (1995) provide a superb detailed analysis of these principles; we will summarize some of the most important visual interface design concepts here.

Visual interfaces should:

- Avoid visual noise and clutter
- Use contrast, similarity, and layering to distinguish and organize elements
- Provide visual structure and flow at each level of organization
- Use cohesive, consistent, and contextually appropriate imagery
- Integrate style and function comprehensively and purposefully

We discuss each of these principles in more detail in the following sections

Avoid visual noise and clutter

Visual noise in interfaces is the result of superfluous visual elements that distract from those visual elements that directly communicate software function and behavior. Imagine trying to hold a conversation in an exceptionally crowded and loud restaurant. It can become impossible to communicate if the atmosphere is too noisy. The same is true for user interfaces. Visual noise can take the form of over-embellished and unnecessarily dimensional elements, overuse of rules and other visual elements to separate controls, insufficient use of white space between controls, and inappropriate or overuse of color, texture, and typography.

Cluttered interfaces attempt to provide an excess of functionality in a constrained space, resulting in controls that visually interfere with each other. Visually baroque, jumbled, or overcrowded screens raise the cognitive load for the user and hamper the speed and accuracy of user attempts at navigation.

In general, interfaces — non-entertainment interfaces, in particular — should use simple geometric forms, minimal contours, and less-saturated colors. Typography should not vary widely in an interface: Typically one or two typefaces in a few sizes are sufficient. When multiple, similar design elements {controls, panes, windows) are required for similar or related logical purpose, they should be quite similar in visual attributes such as shape, size, texture, color, weight, orientation, spacing, and alignment. Elements intended to stand out should be visually contrasted with any regularized elements.

Good visual interfaces, like any good visual design, are visually *efficient.* They make the best use out of the minimal set of visual and functional elements. A popular technique

used by graphic designers is to experiment with the removal of individual elements in order to test their contribution to the clarity of the intended message.

© Copyright Virtual University of Pakistan

Pilot and poet Antoine de Saint Exupery once expressed, "Perfection is attained not when there is no longer anything to add, but when there is no longer anything to take away." As you create your interfaces, you should constantly be looking to simplify visually. The more useful work a visual element can accomplish, while retaining clarity, the better. As Albert Einstein suggested, things should be as simple as possible, but no simpler

Another related concept is that of leverage, using elements in an interface for multiple, related purposes. A good example is a visual symbol that communicates the type of an object in a list, which when clicked on also opens a properties dialog for that object type. The interface could include a separate control for launching the properties display, but the economical and logical solution is to combine it with the type marker. In general, interaction designers, not visual designers, are best suited to tackle the assignment of multiple functions to visual elements. Such mapping of elements requires significant insight into the behavior of users in context, the behavior of the software, and programming issues.

Use contrast and layering to distinguish and organize elements

There are two needs addressed by providing contrast in the elements of an interface. The first is to provide visual contrast between active, manipulable elements of the interface, and passive, non-manipulable visual elements. The second is to provide contrast between different logical sets of active elements to better communicate their distinct functions. Unintentional or ambiguous use of contrast should be avoided, as user confusion almost certainly results. Proper use of contrast will result in visual patterns that users register and remember, allowing them to orient themselves much more rapidly. Contrast also provides a gross means of indicating the most or least important elements in an interface's visual hierarchy. In other words, contrast is a tool for the communication of function and behavior.

DIMENSIONAL, TONAL, AND SPATIAL CONTRAST

The manipulable controls of an interface should visually stand out from non-manipulable regions. Use of pseudo-3D to give the feel of a manual affordance is perhaps the most effective form of contrast for controls. Typically, buttons and other items to be clicked or dragged are given a raised look, whereas data entry areas like text fields are given indented looks. These techniques provide dimensional contrast.

In addition to the dimensionality of affordance, *hue, saturation,* or *value* (brightness) can be varied to distinguish controls from the background or to group controls logically. When using such tonal contrast, you should in most cases vary along a single "axis" — hue or saturation or value, but not all at once. Also, be aware that contrasting by hue runs the risk of disenfranchising individuals with color perception problems; saturation or brightness is probably a safer alternative. In grayscale displays, tonal contrast by value is the only choice the designer has. Depending on the context, tonal contrast of either the controls, of the background area the controls rest on, or of both may be appropriate.

Spatial contrast is another way of making logical distinctions between controls and data entry areas. By positioning related elements together spatially, you help make clear to the user what tasks relate to each other. Good grouping *by position* takes into account the order of tasks and subtasks and how the eye scans the screen (left to right in most Western countries, and generally from top to bottom), which we discuss more in a following section. Shape is also an important form of contrast: Check boxes, for example, are square, whereas radio buttons

© Copyright Virtual University of Pakistan

are round — a design decision not made by accident. Another type of spatial contrast is *orientation:* up, down, left, right, and the angles in between. Icons on the Mac and in Windows provide subtle orientation cues: Document icons are more vertical, folders more horizontal, and application icons, at least on the original Mac, had a diagonal component. Contrast *of size* is also useful, particularly in the display of quantitative information, as it easily invites comparison. We talk more about information design later in this chapter. Contrast in size is also useful when considering the relative sizes of titles and labels, as well as the relative sizes of modular regions of an interface grid. Size, in these cases, can relate to broadness of scope, to importance, and to frequency of use. Again, as with tonal contrast, sticking to a single "axis" of variation is best with spatial contrast.

LAYERING

Interfaces can be organized by layering visual cues in individual elements or in the background on which the active elements rest. Several visual attributes control the perception of layers. Color affects perception of layering: Dark, cool, de-saturated colors recede, whereas light, warm, saturated colors advance. Size also affects layering: Large elements advance whereas small elements tend to recede. Positionally overlapping elements are perhaps the most straightforward examples of visual layering.

To layer elements effectively, you must use a minimum amount of contrast to maintain close similarity between the items you wish to associate in a layer on the screen. After you have decided what the groups are and how to best communicate about them visually, you can begin to adjust the contrast of the groups to make them more or less prominent in the display, according to their importance in context. Maximize differences between layers, but minimize differences between items within a layer.

FIGURE AND GROUND

One side effect of the way humans visually perceive patterns is the tension between the figure, the visual elements that should be the focus of the users attention, and the ground, the background context upon which the figure appears. People tend to perceive light objects as the figure and dark objects as the ground. Figure and ground need to be integrated in a successful design: Poorly positioned and scaled figure elements may end up emphasizing the ground. Well-integrated designs feature figure and ground that are about equal in their scale and visual weight and in which the figure is centered on the ground.

THE SQUINT TEST

A good way to help ensure that a visual interface design employs contrast effectively is to use what graphic designers refer to as the squint test. Close one eye and squint at the screen with the other eye in order to see which elements pop out and which are fuzzy, which items seem to group together, and whether figure or ground seem dominant. Other tests include viewing the design through a mirror (the mirror test) and looking at the design upside down to uncover imbalances in the design. Changing your perspective can often uncover previously undetected issues in layout and composition.

© Copyright Virtual University of Pakistan

Lecture 39.

Behavior & Form – Part VII

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

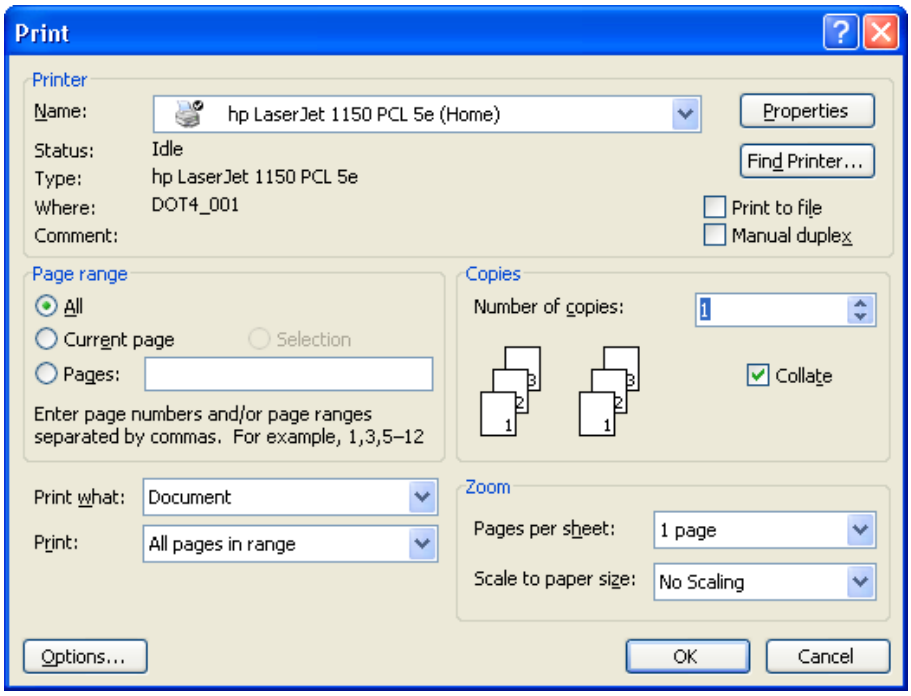
- Understand the Principles of visual interface design

39.1 Provide visual structure and flow at each level of organization

Your interfaces are most likely going to be composed of visual and behavioral elements used in f groups, which are then grouped together into panes, which then may, in turn, be grouped into screens or pages. This grouping can be by position (or proximity), by alignment, by color (value, hue, temperature, saturation), by texture, by size, or by shape. There may be several such levels of structure in a sovereign application, and so it is critical that you maintain a clear visual structure so that the user can easily navigate from one part of your interface to another, as his workflow requires. The rest of this section describes several important attributes that help define a crisp visual structure.

Alignment; grids, and the user's logical path

Alignment of visual elements is one of the key ways that designers can help users experienced product in an organized, systematic way. Grouped elements should be aligned both horizontally and vertically (see Figure).

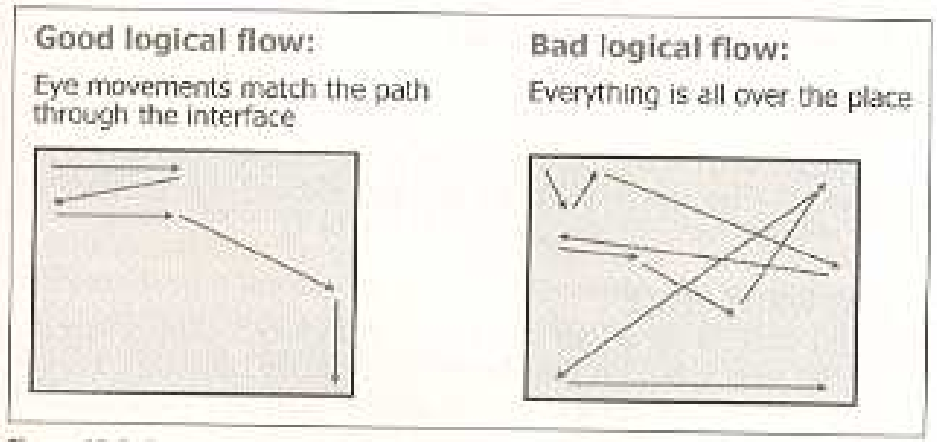


© Copyright Virtual University of Pakistan

In particular, designers should take care to

- **Align labels**. Labels for controls stacked vertically should be aligned with each other; left-justification is easier for users to scan than right justification, although the latter may look visually cleaner — if the input forms are the same size. (Otherwise, you get a Christmas tree, ragged-edge effect on the left and right.)
- **Align within a set of controls**. A related group of check boxes, radio buttons, or text , fields should be aligned according to a regular grid.
- **Align across controls.** Aligned controls (as described previously) that are grouped together with other aligned controls should all follow the same grid.
- **Follow a regular grid structure** for larger-scale element groups, panes, and screens, as well as for smaller grouping of controls.

A grid structure is particularly important for defining an interface with several levels of visual or functional complexity. After interaction designers have defined the overall framework for the application and its elements, visual interface designers should help regularize the layout into a grid structure that properly emphasizes top-level elements and structures but still provides room for lower level or less important controls. The most important thing to remember about grids is that simple is better. If the atomic grid unit is too small, the grid will become unrecognizable in its complexity. Ambiguity and complexity are the enemies of good design. Clear, simple grids help combat ambiguity.



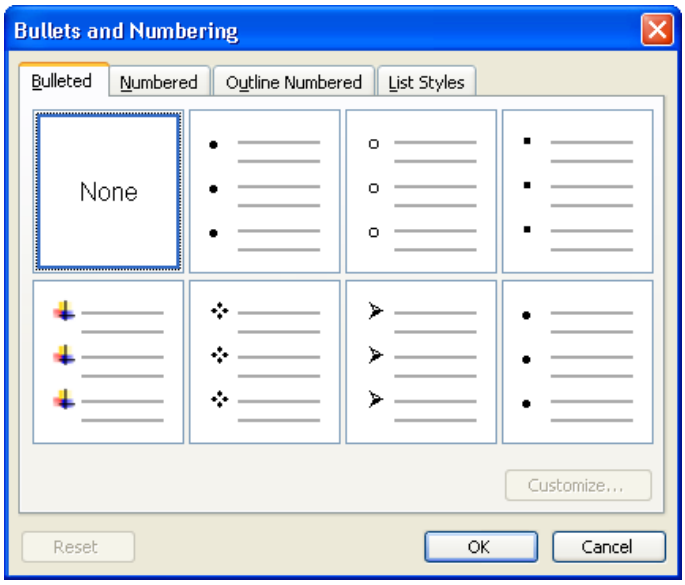
The layout, although conforming to the grid, must also properly mirror the user's logical path through the application, taking into account the fact that (in Western countries) the eye will move from top to bottom and left to right (see Figure).

SYMMETRY AND BALANCE

Symmetry[1] is a useful tool in organizing interfaces from the standpoint of providing visual balance. Interfaces that don't employ symmetry tend to look unbalanced, as if they are going to topple over to one side. Experienced visual designers are adept at achieving asymmetrical balance by controlling the visual weight ot individual elements much as you might balance children of different weights on a seesaw. Asymmetrical design is difficult to achieve in the context of user interfaces because of the high premium placed on white space by screen real-estate constraints. The squint test, the mirror test, and the upside down test are again useful for seeing whether a display looks lopsided.

© Copyright Virtual University of Pakistan

Two types of symmetry are most often employed in interfaces: vertical axial symmetry (symmetry along a vertical line, usually drawn down the middle of a group of elements) or diagonal axial symmetry (symmetry along a diagonal line). Most typical dialog boxes exhibit one or the other of these symmetries — most frequently diagonal symmetry (see Figure).



Sovereign applications typically won't exhibit such symmetry' at the top level (they achieve balance through a well-designed grid), but elements within a well-designed sovereign interface will almost certainly exhibit use of symmetry' to some degree (see Figure).



© Copyright Virtual University of Pakistan

SPATIAL HARMONY AND WHITE SPACE

Spatial harmony considers the interface (or at least each screen) as a whole. Designers have discovered that certain proportions seem to be more pleasing than others to the human eye. The best known of these is the (Golden Section ratio, discovered in antiquity — likely by the Greeks —and probably coined by Leonardo Da Vinci. Unfortunately, for the time being, most computer monitors have a ratio of 1.33:1, which puts visual designers at a bit of a disadvantage when laying out full-screen, sovereign applications. Nonetheless, the understanding of such ratios makes a big difference in developing comfortable layouts for user interfaces.

Proper dimensioning of interface functional regions adds to spatial harmony, as does a proper amount of white space between elements and surrounding element groups. Just as well-designed books enforce proper margins and spacing between paragraphs, figures, and captions, the same kind of visual attention is critical to designing an interface that does not seem cramped or uncomfortable. Especially in the case of sovereign applications, which users will be inhabiting for many hours at a time, it is critical to get proportions right. The last thing you want is for your user to feel uncomfortable and irritated every time she uses your product or service. The key is to be decisive in your layout. Almost a square is no good. Almost a double square is also no good. Make your proportions bold, crisp, and exact.

Use cohesive, consistent, and contextually appropriate imagery

Use of icons and other illustrative elements can help users understand an interface, or if poorly executed, can irritate, confuse, or insult. It is important that designers understand both what the program needs to communicate to users and how users think about what must be communicated. A good understanding of personas and their mental models should provide a solid foundation for both textual and visual language used in an interface. Cultural issues are also important. Designers should be aware of different meanings for colors in different cultures (red is not a warning color in China), for gestures (thumbs up is a terrible insult in Turkey), and for symbols (an octagonal shape means a stop in the US, but not in many other countries). Also, be aware of domain-specific color coding. Yellow means radiation in a hospital. Red usually means something life-threatening. Make sure you understand the visual language of your users' domains and environments before forging ahead.

Visual elements should also be part of a cohesive and globally applied visual language. This means that similar elements should share visual attributes, such as how they are positioned, their size, line weight, and overall style, contrasting only what is important to differentiate their meaning. The idea is to create a system of elements that integrate together to form a cohesive whole. A design that achieves this seems to fit together perfectly; nothing looks stuck on at the last minute.

FUNCTION-ORIENTED ICON

Designing icons to represent functions or operations performed on objects leads to interesting challenges. The most significant challenge is to represent an abstract concept in iconic, visual language. In these cases, it is best to rely on idioms rather than force a concrete representation where none makes sense and to consider the addition of ToolTips or text labels. For more obviously concrete functions, some guidelines apply:

- Represent both the action and an object acted upon to improve comprehension. Nouns and verbs are easier to comprehend together than verbs alone (for example, for a Cut command, representing a document with an X through it may be more readily understood than a more metaphorical image of a pair of scissors).

© Copyright Virtual University of Pakistan

- Beware of metaphors and representations that may not have the intended meanings for your target audience.
- Group related functions visually to provide context, either spatially or, if this is not appropriate, using color or other common visual themes.
- Keep icons simple; avoid excessive visual detail.
- Reuse elements when possible, so users need to learn them only once.

ASSOCIATING VISUAL SYMBOLS TO OBJECTS

Creating unique symbols for types of objects in the interface supports user recognition. These symbols can't always be representational or metaphoric — they are thus often idiomatic. Such visual markers help the user navigate to appropriate objects faster than text labels alone would allow. To establish the connection between symbol and object, use the symbol wherever the object is represented on the screen.

RENDERING ICONS AND VISUAL SYMBOLS

Especially as the graphics capabilities of color screens increase, it is tempting to render icons and visuals with ever-increasing detail, producing an almost photographic quality. However, this trend does not ultimately serve user goals, especially in productivity applications. Icons should remain, simple and schematic, minimizing the number of colors and shades and retaining a modest size. Both Windows XP and Mac OS X have recently taken the step towards more fully rendered icons (OS X more so, with its 128x128 pixel, nearly photographic icons). Although such icons may look great, they draw undue attention to themselves and render poorly at small sizes, meaning that hey must necessarily take up extra real estate to be legible. They also encourage a lack of visual cohesion in the interface because only a small number of functions (mostly those related to hardware) can be adequately represented with such concrete photo-realistic images. Photographic cons are like all-capitalized text; the differences between icons aren't sharp and easy to distinguish, so we get lost in the complexity. The Mac OS X Aqua interface is filled with photo-realistic touches that ultimately distract. None of this serves the user particularly well.

VISUALIZING BEHAVIORS

Instead of using words alone to describe the results of interface functions (or worse, not giving any description at all), use visual elements *to show* the user what the results will be. Don't confuse this with use of icons on control affordances. Rather, in addition to using text to communicate a setting or state, render an illustrative picture or diagram that communicates the *behavior*. Although visualization often consumes more space, its capability to clearly communicate is well worth the pixels. In recent years, Microsoft has discovered this fact, and the dialog boxes in Windows Word, for example, have begun to bristle with visualizations of their meaning in addition to the textual controls. Photoshop and other image-manipulation applications have long shown thumbnail previews of the results of visual processing operations The Word Page Setup dialog box offers an image labeled Preview. This is an output-only control, showing a miniature view of what the page will look like with the current margin settings on the dialog. Most users have trouble visualizing what a 1.2 inch left margin looks like. The Preview control shows them. Microsoft could go one better by allowing input on the Preview control in addition to
output. Drag the left margin of the picture and watch the numeric value in the corresponding spinner ratchet up and down.

© Copyright Virtual University of Pakistan

The associated text field is still important — you can't just replace it with the visual one. The text shows the precise values of the settings, whereas the visual control accurately portrays the look of the resulting page.

Integrate style and function comprehensively and purposefully

When designers choose to apply stylistic elements to an interface, it must be from a global perspective. Every aspect of the interface must be considered from a stylistic point of view, not simply individual controls or other visual elements. You do not want your interface to seem as though someone applied a quick coat of paint. Rather you need to make sure that the functional aspects of your program's visual interface design are in complete harmony with the visual brand of your product. Your program's behavior is part of its brand, and your user's experience with your product should reflect the proper balance of form, content, and behavior.

FORM VERSUS FUNCTION

Although visual style is a tempting diversion for many visual designers, use of stylized visual elements needs to be carefully controlled within an interface — particularly when designing for sovereign applications. Designers must be careful not to affect the basic shape, visual behavior, and visual affordance of controls in the effort to adapt them to a visual style. The point is to be aware of the value each element provides. There's nothing wrong with an element that adds style, as long as it accomplishes what you intend and doesn't interfere with the meaning of the interface or the user's ability to interact with it.

That said, educational and entertainment applications, especially those designed for children, leave room for a bit more stylistic experimentation. The visual experience of the interface and content are part of the enjoyment of these applications, and a greater argument can also be made for thematic relationships between controls and content. Even in these cases, however, basic affordances should be preserved so that users can, in fact, reach the content easily.

BRANDING AND THE USER INTERFACE

Most successful companies make a significant investment in building brand equity. A company that cultivates substantial brand equity can command a price premium for its products, while encouraging greater customer loyalty. Brands indicate the positive characteristics of the product and suggest discrimination and taste in the user.

In its most basic sense, brand value is the sum of all the interactions people have with a given company. Because an increasing number of these interactions are occurring through technology-based channels, it should be no surprise that the emphasis placed on branding user interfaces is heater than ever. If the goal is consistently positive customer interactions, the verbal, visual, and behavioral brand messages must be consistent.

Although companies have been considering the implications of branding as it relates to traditional marketing and communication channels for some time now, many companies are just beginning to address branding in terms of the user interface. In order to understand branding in the context of the user interface, it can be helpful to think about it from two perspectives: the first impression and the long-term relationship.

Just as with interpersonal relationships, first impressions of a user interface can be exceedingly important. The first five-minute experience is the foundation that long-term relationships are built upon. To ensure a successful first five-minute experience, a user interface must clearly and immediately communicate the brand. Visual design, typically, plays one of the most significant roles in managing first impressions largely through color

© Copyright Virtual University of Pakistan

and image. By selecting a color palette and image style for your user interface that supports the brand, you go a long way toward leveraging the equity of that brand in the form of a positive first impression.
After people have developed a first impression, they begin to assess whether the behavior of the interface is consistent with its appearance. You build brand equity and long-term customer relationships by delivering on the promises made during the first impression. Interaction design and the control of behavior are often the best ways to keep the promises that visual branding makes to users.

39.2 Principles of Visual Information Design

Like visual interface design, visual information design also has many principles that the prospective designer can use to his advantage. Information design guru Edward Tufte asserts that good visual design is "clear thinking made visible," and that good visual design is achieved through an understanding of the viewer's "cognitive task" (goal) and a set of design principles.
Tufte claims that there are two important problems in information design:

1. It is difficult to display multidimensional information (information with more than two variables) on a two-dimensional surface.
2. The resolution of the display surface is often not high enough to display dense information. Computers present a particular challenge — although they can add motion and interactivity, computer displays have a low information density compared to that of paper.

Interaction and visual interface designers may not be able to escape the limitations of 2D screens or overcome the problems of low-resolution displays. However, some universal design principles — indifferent to language, culture, or time — help maximize the effectiveness of any information display, whether on paper or digital media.
In his beautifully executed volume, *The Visual Display of Quantitative Information* (1983), Tufte introduces seven Grand Principles, which we briefly discuss in the following sections as they relate specifically to digital interfaces and content.
Visually displayed information should, according to Tufte

1. Enforce visual comparisons
2. Show causality
3. Show multiple variables
4. Integrate text, graphics, and data in one display
5. Ensure the quality, relevance, and integrity of the content
6. Show things adjacently in space, not stacked in time
7. Not de-quantify quantifiable data

We will briefly discuss each of these principles as they apply to the information design of software-enabled media.

Enforce visual comparisons

You should provide a means for users to compare related variables and trends or to compare before and after scenarios. Comparison provides a context that makes the information more valuable and more comprehensible to users. Adobe Photoshop, along with many other graphics tools, makes frequent use of previews, which allow users to easily

© Copyright Virtual University of Pakistan

achieve before and after comparisons interactively (see Figure A as well as Figures B and C).

Show causality

Within information graphics, clarify cause and effect. In his books, Tufte provides the classic example of the space shuttle Challenger disaster, which could have been averted if charts prepared by NASA scientists had been organized to more clearly present the relationship between air temperature at launch and severity of 0-ring failure. In interactive interfaces, modeless visual feedback should be employed to inform users of the potential consequences of their actions or to provide hints on how to perform actions.



Figure A

Show multiple variables

Data displays that provide information on multiple, related variables should be able to display them all simultaneously without sacrificing clarity. In an interactive display, the user should be able to selectively turn off and on the variables to make comparisons easier and correlations (causality) clearer. Figure B shows an example of an interactive display that permits manipulation of multiple variables.

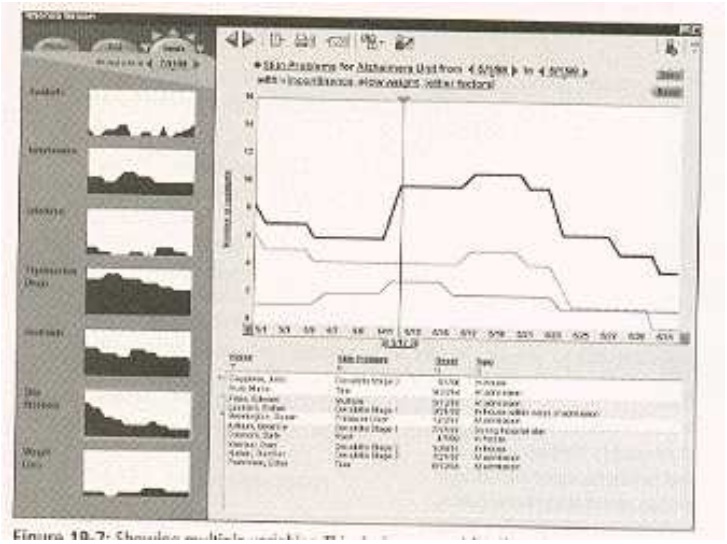


Figure B

© Copyright Virtual University of Pakistan

Integrate text, graphics, and data in one display

Diagrams that require separate keys or legends to decode are less effective and require more cognitive processing on the part of users. Reading and deciphering diagram legends is yet another form of navigation-related excise. Users must move their focus back and forth between diagram and legend and then reconcile the two in their minds. Figure C shows an interactive example (integrates text, graphics, and data, as well as input and output: a highly efficient combination for users.

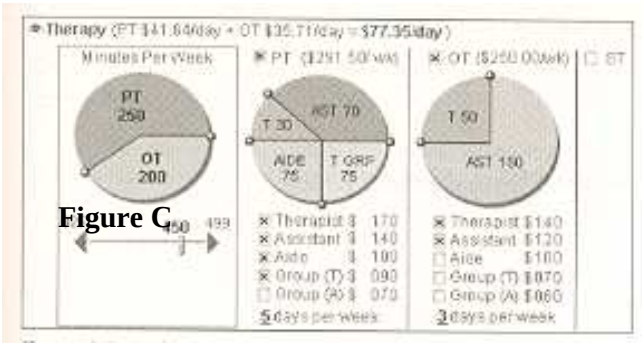


Figure C

Ensure the quality, relevance, and integrity of the content

Don't show information simply because it's technically possible to do so. Make sure that any information you display will help your users achieve particular goals that are relevant to their context. Unreliable or otherwise poor-quality information will damage the trust you must build with users through your product's content, behavior, and visual brand.

Show things adjacently in space, not stacked in time

If you are showing changes over time, it's much easier for users to understand the changes if they are shown adjacently in space, rather than superimposed on one another. Cartoon strips are a good example of showing flow and change over time arranged adjacently in space.
Of course, this advice applies to static information displays; in software, animation can be used even more effectively to show change over time, as long as technical issues (such as memory constraints or connection speed over the Internet) don't come into play.

Don't de-quantify quantifiable data

Although you may want to use graphs and charts to make perception of trends and other quantitative information easier to grasp, you should not abandon the display of the numbers themselves. For example, in the Windows Disk Properties dialog, a pie chart is displayed to give users a rough idea of their tree disk space, but the numbers of kilobytes free and used are also displayed in numerical form.

39.3 Use of Text and Color in Visual Interfaces

Text and color are both becoming indispensable elements of the visual language of user interfaces (text always has been). This section discusses some useful visual principles concerning the use of these two important visual tools.

© Copyright Virtual University of Pakistan

Use of text

Humans process visual information more easily than they do textual information, which means that navigation by visual elements is faster than navigation by textual elements. For navigation purposes, text words are best considered as visual elements. They should, therefore, be short, easily recognized, and easily remembered.

Text forms a recognizable shape that our brains categorize as a visual object. Each word has a recognizable shape, which is why WORDS TYPED IN ALL CAPITAL LETTERS ARE HARDER TO READ than upper/lowercase — the familiar pattern-matching hints are absent in capitalized words, so we must pay much closer attention to decipher what is written. Avoid using all caps in your interfaces.

Recognizing words is also different from reading, where we consciously scan the individual words and interpret their meaning in context. Interfaces should try to minimize the amount of text that must be read in order to navigate the interface successfully: After the user has navigated to something interesting, he should be able to read in detail if appropriate. Using visual objects to provide context facilitates navigation with minimal reading.

Our brains can rapidly differentiate objects in an interface if we represent what objects are by using visual symbols and idioms. After we have visually identified the type of object we are interested in, we can read the text to distinguish which particular object we are looking at. In this scheme, we don't need to read about types of objects we are not interested in, thus speeding navigation and eliminating excise. The accompanying text only comes into play after we have decided that it is important.

When text must be read in interfaces, some guidelines apply:

- Make sure that the text is in high contrast with the background and do not use conflicting colors that may affect readability.
- Choose an appropriate typeface and point size. Point sizes less than 10 are difficult to read. For brief text, such as on a label or brief instruction, a crisp sans-serif font, like Arial, is appropriate; for paragraphs of text, a serif font, like Times, is more appropriate.
- Phrase your text to make it understandable by using the least number of words necessary to clearly convey meaning. Phrase clearly, and avoid abbreviation. If you must abbreviate, use standard abbreviations.

Use of color

Color is an important part of most visual interfaces whose technology can support it. In these days of ubiquitous color LCDs, users have begun to expect color screens even in devices like PDAs and phones. However, color is much more than a marketing checklist item; it is a powerful information design and visual interface design tool that can be used to great effect, or just as easily abused.

Color communicates as part of the visual language of an interface, and users will impart meaning to its use. For non-entertainment, sovereign applications in particular, color should integrate well into the other elements of the visual language: symbols and icons, text, and the spatial relationships they maintain in the interface. Color, when used appropriately, serves the following purposes in visual interface design:

- Color draws attention. Color is an important element in rich visual feedback, and consistent use of it to highlight important information provides an important channel of communication.
- Color improves navigation and scanning speed. Consistent use of color in signposts can help users quickly navigate and home in on information they are looking for.

© Copyright Virtual University of Pakistan

- Color shows relationships. Color can provide a means of grouping or relating objects together.

Misuse of color

There are a few ways that color can be misused in an interface if one is not careful. The most common of these misuses are as follows:

- **Too many colors**. A study by Human Factors International indicated that one color significantly reduced search time. Adding additional colors provides less value, and at seven or more, search performance degraded significantly. It isn't unreasonable to suspect a similar pattern in any kind of interface navigation.
- **Use of complementary colors.** Complementary colors are the inverse of each other in color computation. These colors, when put adjacent to each other or when used together as figure and ground, create perceptual artifacts that are difficult to perceive correctly or focus on. A similar effect is the result of chromostereopsis, in which colors v on the extreme ends of the spectrum "vibrate" when placed adjacently. Red text on a blue background (or vice versa) is extremely difficult to read.
- **Excessive saturation**. Highly saturated colors tend look garish and draw too much attention. When multiple saturated colors are used together, chromostereopsis and other perceptual artifacts often occur.
- **Inadequate contrast**. When figure colors differ from background colors only in hue, but not in saturation or value (brightness), they become difficult to perceive. Figure and ground should vary in brightness or saturation, in addition to hue, and color text on color backgrounds should also be avoided when possible.
- **Inadequate attention to color impairment**. Roughly ten percent of the male population has some degree of color-blindness. Thus care should be taken when using red and green hues (in particular) to communicate important information. Any colors used to communicate should also vary by saturation or brightness to
- distinguish them from each other. If a grayscale conversion of your color
- palette is easily distinguishable, colorblind users should be able to distinguish the color version.

39.4 Consistency and Standards

Many in-house usability organizations view themselves, among other things, as the gatekeepers of consistency in digital product design. Consistency implies a similar look, feel, and behavior across the various modules of a software product, and this is sometimes extended to apply across all the products a vendor sells. For at-large software vendors, such as Macromedia and Adobe, who regularly acquire new software titles from smaller vendors, the branding concerns of consistency take on a particular urgency. It is obviously in their best interests to make acquired software look as though it belongs, as a first-class offering, alongside products developed in-house. Beyond this, both Apple and Microsoft have an interest in encouraging their own and third-party developers to create applications that have the look and feel of the OS platform on which the program is being run, so that the user perceives their respective platforms as providing a seamless and comfortable user experience.

Benefits of interface standards

User interface standards provide benefits that address these issues, although they come at a price. Standards provide benefits to users when executed appropriately. According to Jakob Nielsen (1993), relying on a single interface standard improves users' ability to quickly learn interfaces and enhances their productivity by raising throughput and reducing errors. These

© Copyright Virtual University of Pakistan

benefits accrue because users are more readily able to predict program behavior based on past experience with other parts of the interface, or with other applications following similar standards.

At the same time, interface standards also benefit software vendors. Customer training and technical support costs are reduced because the consistency that standards bring improves ease of use and learning. Development time and effort are also reduced because formal interface standards provide ready-made decisions on the rendering of the interface that development teams would otherwise be forced to debate during project meetings. Finally, good standards can lead to reduced maintenance costs and improved reuse of design and code.

Risks of interface standards

The primary risk of any standard is that the product that follows it is only as good as the standard itself. Great care must be made in developing the standard in the first place to make sure, as Nielsen says, that the standard specifies a truly usable interface, and that it is usable by the *developers* who must build the interface according to its specifications.

It is also risky to see interface standards as a panacea for good interfaces. Most interface standards emphasize the *syntax* of the interface, its visual look and feel, but say little about deeper behaviors of the interface or about its higher-level logical and organizational structure. There is a good reason for this: A general interface standard has no knowledge of context incorporated into its formalizations. It takes into account no

specific user behaviors and usage patterns within a context, but rather focuses on general issues of human perception and cognition and, sometimes, visual branding as well. These concerns are important, but they are presentation details, not the interaction framework upon which such rules hang.

Standards, guidelines, and rules of thumb

Although standards are unarguably useful, they need to evolve as technology and our understanding of users and their goals evolve. Some practitioners and programmers invoke Apple's or Microsoft's user interface standards as if they were delivered from Mt. Sinai on a tablet. Both companies publish user interface standards, but both companies also freely and frequently violate them and update the guidelines post facto. When Microsoft proposes an interface standard, it has no qualms about changing it for something better in the next version. This is only natural — interface design is still in its infancy, and it is wrongheaded to think that there is benefit in standards that stifle true innovation. In some respects, Apple's dramatic visual shift from'OS 9 to OS X has helped to dispel the notion among the Mac faithful that interface standards are etched in granite.

The original Macintosh was a spectacular achievement precisely because it transcended all Apple's previous platforms and standards. Conversely, much of the strength of the Mac came from the fact that vendors followed Apple's lead and made their interfaces look, work, and act alike. Similarly, many successful Windows programs are unabashedly modeled after Word, Excel, and Outlook.

interface standards are thus most appropriately treated as detailed *guidelines* or *rules of thumb*. Following interface guidelines too rigidly or without careful consideration of the needs of users in context can result in force-fitting an application's interface into an inappropriate interaction model.

© Copyright Virtual University of Pakistan

When to violate guidelines

So, what should we make of interface guidelines? Instead of asking if we should follow standards, it is more useful to ask: When should we violate standards? The answer is when, and only when, we have a very good reason.

But what constitutes a very good reason? Is it when a new idiom is measurably better? Usually, this sort of measurement can be quite elusive because it rarely reduces to a quantifiable factor alone. The best answer is: When an idiom is clearly seen to be significantly better by most people in the target user audience (your personas) who try it, there's a good reason to keep it in the interlace. This is how the toolbar came into existence, along with outline views, tabs, and many other idioms. Researchers may have been examining these artifacts in the lab, but it was their useful presence in real-world software that confirmed the success.

Your reasons for diverging from guidelines may ultimately not prove to be good enough and your product may suffer. But you and other designers will learn from the mistake. This is what Christopher Alexander (1964) calls the "unselfconscious process," an indigenous and unexamined process of slow and tiny forward increments as individuals attempt to improve solutions. New idioms (as well as new uses for old idioms) pose a risk, which is why careful, goal-directed design and appropriate testing with real users in real working conditions are so important.

Consistency and standards across applications

Using standards or guidelines has special challenges when a company that sells multiple software titles decides that all its various products must be completely consistent from a user-interface perspective.

From the perspective of visual branding, as discussed earlier, this makes a great deal of sense, although there are some intricacies. If an analysis of personas and markets indicates that there is little overlap between the users of two distinct products and that their goals and needs are also quite distinct, you might question whether it makes more sense to develop two visual brands that speak specifically to these different customers, rather than using a single, less-targeted look. When it comes to the behavior of the software, these issues become even more urgent. A single standard might be important if customers will be using the products together as a suite. But even in this case, should a graphics-oriented presentation application, like PowerPoint, share an interface structure with a text processor like Word? Microsoft's intentions were good, but it went a little too far enforcing global style guides. PowerPoint doesn't gain much from having a similar menu structure to Excel and Word, and it loses quite a bit in ease-of-use by conforming to an alien structure that diverges from the user's mental models. On the other hand, the designers did draw the line somewhere, and PowerPoint does have a slide-sorter display, an interface unique to that application.

Designers, then, should bear in mind that consistency doesn't imply rigidity, especially where it isn't appropriate. Interface and interaction style guidelines need to grow and evolve like the software they help describe. Sometimes you must bend the rules to best serve your users and their goals (and sometimes even your company's goals). When this has to happen, try to make changes and additions that are compatible with standards. The spirit of the law, not the letter of the law, should be your guide.

© Copyright Virtual University of Pakistan

Lecture 40.

Observing User

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Discuss the benefits and challenges of different types of observation.
- Discuss how to collect, analyze and present data from observational evaluation.

Observation involves watching and listening to users. Observing users interacting with software, even casual observing, can tell you an enormous amount about what they do, the context in which they do it, how well technology supports them, and what other support is needed. In this lecture we describe how to observe and do ethnography and discuss their role in evaluation.

User can be observed in controlled laboratory-like conditions, as in usability testing, or in the natural environments in which the products are used—i.e., the field. How the observation is done depends on why it is being done and the approach adopted. There is a variety of structured, less structured, and descriptive observation techniques for evaluators to choose from. Which they select and how their findings are interpreted will depend upon the evaluation goals, the specific questions being addressed, and practical constraints.

40.1 What and when to observe

Observing is useful at any time during product development. Early in design, observation helps designers understand users' needs. Other types of observation are done later to examine whether the developing prototype meets users' needs.

Depending on the type of study, evaluators may be onlookers, participant observers, or ethnographers. The degree of immersion that evaluators adopt varies across a broad outsider-insider spectrum. Where a particular study falls along this spectrum depends on its goal and on the practical and ethical issues that constrain and shape it.

40.2 How to observe

The same basic data-collection tools are used for laboratory and field studies (i.e., direct observation, taking notes, collecting video, etc.) but the way in which they are used is different. In the laboratory the emphasis is on the details of what individuals do, while in the field the context is important and the focus is on how people interact with each other, the technology, and their environment. Furthermore, the equipment in the laboratory is usually set up in advance and is relatively static whereas in the field it usually must be moved around. In this section we discuss how to observe, and then examine the practicalities and compare data-collection tools.

In controlled environments

The role of the observer is to first collect and then make sense of the stream of data on video, audiotapes, or notes made while watching users in a controlled

© Copyright Virtual University of Pakistan

environment. Many practical issues have to be thought about in advance, including the following.

- It is necessary to decide where users will be located so that the equipment can be set up. Many usability laboratories, for example, have two or three wall-mounted, adjustable cameras to record users'

activities while they work on test tasks. One camera might record facial expressions, another might focus on mouse and keyboard activity, and another might record a broad view of the participant and capture body language. The stream of data from the cameras is fed into a video editing and analysis suite where it is annotated and partially edited. Another form of data that can be collected is an interaction log. This records all the user's key presses. Mobile usability laboratories, as the name suggests, are intended to be moved around, but the equipment can be bulky. Usually it is taken to a customer's site where a temporary laboratory environment is created.

- The equipment needs testing to make sure that it is set up and works as expected, e.g., it is advisable that the audio is set at the right level to record the user's voice.
- An informed consent form should be available for users to read and sign at the beginning of the study. A script is also needed to guide how users are greeted, and to tell them the goals of the study, how long it will last, and to explain their rights. It is also important to make users feel comfortable and at ease.

In the field

Whether the observer sets out to be an outsider or an insider, events in the field can be complex and rapidly changing. There is a lot for evaluators to think about, so many experts have a framework to structure and focus their observation. Ilk framework can be quite simple. For example, this is a practitioner's framework that focuses on just three easy-to-remember items to look for:

- The Person. Who is using the technology at any particular time?
- The Place. Where are they using it?
- The Thing. What are they doing with it?

Frameworks like the one above help observers to keep their goals and questions in sight. Experienced observers may, however, prefer more detailed frame works, such as the one suggested by Goetz and LeCompte (19X4) below, which encourages observers to pay greater attention to the context of events, the people and the technology:
Who is present? How would you characterize them? What is their role?
What is happening? What are people doing and saying and how are they behaving? Does any of this behavior appear routine? What is their tone and body language?
When does the activity occur? How is it related to other activities?
Where is it happening? Do physical conditions play a role?
Where is it happening? What precipitated the event or interaction? Do people have different perspectives?

© Copyright Virtual University of Pakistan

How is the activity organized? What rules or norms influence behavior?
Colin Kobson (1993) suggests a slightly longer but similar set of items:

- Space. What is the physical space like and how is it laid out?
- Actors. What are the names and relevant details of the people involved?
- Activities. What are the actors doing and why?
- Objects. What physical objects are present, such as furniture?
- Acts. What are specific individuals doing?
- Events. Is what you observe part of a special event?
- Goals. What are the actors trying to accomplish?
- Feelings. What is the mood of the group and of individuals?

These frameworks are useful not only for providing focus but also for organizing the observation and data-collection activity. Below is a checklist of things to plan before going into the field:

- State the initial study goal and questions clearly.
- Select a framework to guide your activity in the field.
- Decide how to record events—i.e., as notes, on audio, or on video, or using a combination of all three. Make sure you have the appropriate equipment and that it works. You need a suitable notebook and pens. A laptop computer might be useful but could be cumbersome. Although this is called observation, photographs, video, interview transcripts and the like will help to explain what you see and are useful for reporting the story to others.
- Be prepared to go through your notes and other records as soon as possible after each evaluation session to flesh out detail and check ambiguities with other observers or with the people being observed. This should be done routinely because human memory is unreliable. A basic rule is to do it within 24 hours, but sooner is better!
- As you make and review your notes, try to highlight and separate personal opinion from what happens. Also clearly note anything you want to go back to. Data collection and analysis go hand in hand to a large extent in fieldwork.
- Be prepared to re focus your study as you analyze and reflect upon what you see. Having observed for a while, you will start to identify interesting phenomena that seem relevant. Gradually you will sharpen your ideas into questions that guide further observation, either with the same group or with a new but similar group.
- Think about how you will gain the acceptance and trust of those you observe. Adopting a similar style of dress and finding out what interests the group and showing enthusiasm for what they do will help. Allow time to develop relationships. Fixing regular times and venues to meet is also helpful, so everyone knows what to expect. Also, be aware that it will be easier to relate lo some people than others, and it will be tempting to pay attention to those who receive you well, so make sure you attend to everyone in the group.

Think about how to handle sensitive issues, such as negotiating where you can go. For example, imagine you are observing the usability of a portable home communication

© Copyright Virtual University of Pakistan

device. Observing in the living room, study, and kitchen is likely to be acceptable, but bedrooms and bathrooms are probably out of bounds. Take time to check what participants are comfortable with and be accommodating and flexible. Your choice of equipment for data collection will also influence how intrusive you are in people's lives.

- Consider working as a team. This can have several benefits: for instance, you can compare your observations. Alternatively, you can agree to focus on different people or different parts of the context. Working as a team is also likely to generate more reliable data because you can compare notes among different evaluators.
- Consider checking your notes with an informant or members of the group to ensure that you are understanding what is happening and that you are making good interpretations.
- Plan to look at the situation from different perspectives. For example, you may focus on particular activities or people. If the situation has a hierarchical structure, as in many companies, you will get different perspectives from different layers of management—e.g., end-users, marketing, product developers, product managers, etc.

Participant observation and ethnography

Being a participant observer or an ethnographer involves all the practical steps just mentioned, but especially that the evaluator must be accepted into the group. An interesting example of participant observation is provided by Nancy Baym's work (1997) in which she joined an online community interested in soap operas for over a year in order to understand how the community functioned. She told the community what she was doing and offered to share her findings with them. This honest approach gained her their trust, and they offered support and helpful comments. As Baym participated she learned about the community, who the key characters were, how people interacted, their values, and the types of discussions that were generated. She kept all the messages as data to be referred to later. She also adapted interviewing and questionnaires techniques to collect additional information.

As we said the distinction between ethnography and participant observation is blurred. Some ethnographers believe that ethnography is an open interpretive approach in which evaluators keep an open mind about what they will see. Others such as David Fetterman from Stanford University see a stronger role for a theoretical underpinning: "before asking the first question in the field the ethnographer begins with a problem, a theory or model, a research design, specific data collection techniques, tools for analysis, and a specific writing style" (Fetterman. 1998. p. 1). This may sound as if ethnographers have biases, but by making assumptions explicit and moving between different perspectives, biases are at least reduced. Ethnographic study allows *multiple* interpretations of reality; it is *interpretvisit.* Data collection and analysis often occur simultaneously in ethnography, with analysis happening at many different levels throughout the

study. The question being investigated is refined as more understanding about the situation is gained.

The checklist below (Fetterman. 1998) for doing ethnography is similar to the general list just mentioned:

Identify a problem or goal and then ask good questions to be answered by the study, which may or may not invoke theory depending on your philosophy of ethnography. The observation framework such as those mentioned above can help to focus the study and stimulate questions.

© Copyright Virtual University of Pakistan

The most important part of fieldwork is just being there to observe, as, questions, and record what is seen and heard. You need to be aware of people's feelings and sensitive to where you should not go.
Collect a variety of data, if possible, such as notes, still pictures, audio and video, and artifacts as appropriate. Interviews are one of the most important data-gathering techniques and can be structured, semi-structured, or open So-called retrospective interviews are used after the fact to check that interpretations are correct.
As you work in the Held, he prepared to move backwards and forwards between the broad picture and specific questions. Look at the situation holistically and then from the perspectives of different stakeholder groups and participants. Early questions arc likely to be broad, but as you get to know the situation ask more specific questions.
Analyze the data using a holistic approach in which observations arc under stood within the broad context—i.e., they are contextualized. To do this, first synthesize your notes, which is best done at the end of each day, and then check with someone from the community that you have described the situation accurately. Analysis is usually iterative, building on ideas with each pass.

40.3 Data Collection

Data collection techniques (i.e., taking notes, audio recording, and video recording) are used individually or in combination and are often supplemented with photos from a still camera. When different kinds of data are collected, evaluators have to coordinate them; this requires additional effort but has the advantage of providing more information and different perspectives. Interaction logging and participant diary studies are also used. Which techniques are used will depend on the context, time available, and the sensitivity of what is being observed. In most settings, audio, photos, and notes will be sufficient. In others it is essential to collect video data so as to observe in detail the intricacies of what is going on.

Notes plus still camera

Taking notes is the least technical way of collecting data, but it can be difficult and tiring to write and observe at the same time. Observers also get bored and the
speed at which they write is limited. Working with another person solves sonic of these problems and provides another perspective. Handwritten notes are flexible in the field but must be transcribed. However, this transcription can be the first step in data analysis, as the evaluator must go through the data and organize it. A laptop computer can be a useful alternative but it is more obtrusive and cumbersome, and its batteries need recharging every few hours. If a record of images is needed, photographs, digital images, or sketches are easily collected.

Audio recording plus still camera

Audio can be a useful alternative to note taking and is less intrusive than video. It allows evaluators to be more mobile than with even the lightest, battery-driven video cameras, and so is very flexible. Tapes, batteries, and the recorder are now relatively inexpensive but there are two main problems with audio recording. One is the lack of a visual record, although this can be dealt with by carrying a small camera. The second drawback is transcribing the data, which can be onerous if the con tents of many hours of recording have to be transcribed: often, however, only sections are needed. Using a headset with foot control makes transcribing less onerous. Many studies do not need this level of

© Copyright Virtual University of Pakistan

detail; instead, evaluators use the recording to remind them about important details and as a source of anecdotes for reports.

Video

Video has the advantage of capturing both visual and audio data but can be intrusive. However, the small, handheld, battery-driven digicams are fairly mobile, inexpensive and are commonly used.

A problem with using video is that attention becomes focused on what is seen through the lens. It is easy to miss other things going on outside of the camera view. When recording in noisy conditions, e.g., in rooms with many computers running or outside when it is windy, the sound may get muffled.

Analysis of video data can be very time-consuming as there is so much to take: note of. Over 100 hours of analysis time for one hour of video recording is common for detailed analyses in which every gesture and utterance is analyzed.

40.4 Indirect observation: tracking users' activities

Sometimes direct observation is not possible because it is obtrusive or evaluators cannot be present over the duration of the study, and so users' activities are tracked indirectly. Diaries and interaction logs are two techniques for doing this. From the records collected evaluators reconstruct what happened and look for usability and user experience problems.

Diaries

Diaries provide a record of what users did, when they did it, and what they thought about their interactions with the technology. They are useful when users are scattered and unreachable in person, as in many Internet and web evaluations. Diaries are inexpensive, require no special equipment or expertise, and are suitable for long-term studies. Templates can also be created online to standardize entry format and enable the data to go straight into a database for analysis. These templates are like those used in open-ended online questionnaires. However, diary studies rely on participants being reliable and remembering to complete them, so incentives are needed and the process has to be straightforward and quick. Another problem is that participants often remember events as being better or worse than they really were, or taking more or less time than they actually did.

Robinson and Godbey (1997) asked participants in their study to record how much time Americans spent on various activities. These diaries were completed at the end of each day and the data was later analyzed to investigate the impact of television on people's lives. In another diary study, Barry Brown and his colleagues from Hewlett Packard collected diaries form 22 people to examine when, how, and why they capture different types of information, such as notes, marks on paper, scenes, sounds, moving images, etc. (Brown, et al.. 2000). The participants were each given a small handheld camera and told to take a picture every time they captured information in any form. The study lasted for seven days and the pictures were used as memory joggers in a subsequent semi-structured interview used to get participants to elaborate on their activities. Three hundred and eighty-one activities were recorded. The pictures provided useful contextual information. From this data the evaluators constructed a framework to inform the design of new digital cameras and handheld scanners.

© Copyright Virtual University of Pakistan

Interaction logging

Interaction logging in which key presses, mouse or other device movements are recorded has been used in usability testing for many years. Collecting this data is usually synchronized with video and audio logs to help evaluators analyze users' behavior and understand how users worked on the tasks they set. Specialist software tools are used to collect and analyze the data. The log is also time-stamped so it can be used to calculate how long a user spends on a particular task or lingered in a certain part of a website or software application.

Explicit counters that record visits to a website were once a familiar sight. Recording the number of visitors to a site can be used to justify maintenance and upgrades to it. For example, if you want to find out whether adding a bulletin board to an e-commerce website increases the number of visits, being able to compare traffic before and after the addition of the bulletin board is useful. You can also track how long people stayed at the site, which areas they visited, where they came from, and where they went next by tracking their Internet Service Provider (I.S.P.) address. For example, in a study of an interactive art museum by researchers at the University of Southern California, server logs were analyzed by tracking visitors in this way (McLaughlin et al., 1999). Records of when people came to the site, what they requested, how long they looked at each page, what browser they were using, and what country they were from, etc., were collected over a seven-month period. The data was analyzed using Webtrends, a commercial analysis tool, and the evaluators discovered that the site was busiest on weekday evenings. In another study that investigated lurking behavior in list server discussion groups, the number of messages posted was compared with list membership over a three-month period to see how lurking behavior differed among groups (Nonnecke and Preece, 2000).

An advantage of logging user activity is that it is unobtrusive, but this also raises ethical concerns that need careful consideration (see the dilemma about observing without being seen). Another advantage is that large volumes of data can be logged automatically. However, powerful tools are needed to explore and analyze this data quantitatively and qualitatively. An increasing number of visualization tools are being developed for this purpose; one example is WebLog, which dynamically shows visits to websites(Hochheiser and Shneiderman, 2000).

40.5 Analyzing, interpreting, and presenting the data

By now you should know that many, indeed most observational evaluations generate a lot of data in the form of notes, sketches, photographs, audio and video records of interviews and events, various artifacts, diaries, and logs. Most observational data is qualitative and analysis often involves interpreting what users were doing or saying by looking for patterns in the data. Sometimes qualitative data is categorized so that it can be quantified and in some studies events are counted.

Dealing with large volumes of data, such as several hours of video, is daunting, which is why it is particularly important to plan observation studies very carefully before starting them. The DECIDE framework suggests identifying goals and questions first before selecting techniques for the study, because the goals and questions help determine which data is collected and how it will be analyzed.

When analyzing any kind of data, the first thing to do is to "eyeball" the data to see what stands out. Are there patterns or significant events? Is there obvious evidence that appears to answer a question or support a theory? Then proceed to analyze it according to the goals and questions. The discussion that follows focuses on three types of data:

© Copyright Virtual University of Pakistan

- *Qualitative data* that is *interpreted* and used to tell "the story" about what was observed.
- *Qualitative data* that is *categorized* using techniques such as content analysis.
- *Quantitative data* that is collected from interaction and video logs and presented as values, tables, charts and graphs and is treated statistically.

Qualitative analysis to tell a story

Much of the power of analyzing descriptive data lies in being able to tell a convincing story, illustrated with powerful examples that help to confirm the main points and will be credible to the development team. It is hard to argue with well-chosen video excerpts of users interacting with technology or anecdotes from transcripts.

To summarize, the main activities involved in working with qualitative data to tell a story are:

- Review the data after each observation session to synthesize and identify key themes and make collections.
- Record the themes in a coherent yet flexible form, with examples. While post-its enable you to move ideas around and group similar ones, they can fall off and get lost and are not easily transported, so capture the main points in another form, either on paper or on a laptop, or make an audio recording.
- Record the date and time of each data analysis session. (The raw data should already be systematically logged with dates.)
- As themes emerge, you may want to check your understanding with the people you observe or your informants.
- Iterate this process until you are sure that your story faithfully represents what you observed and that you have illustrated it with appropriate examples from the data.
- Report your findings to the development team, preferably in an oral presentation as well as in a written report. Reports vary in form, but it is always helpful to have a clear, concise overview of the main findings presented at the beginning.

Quantitative data analysis

Video data collected in usability laboratories is usually annotated as it is observed Small teams of evaluator's watch monitors showing what is being recorded in a control room out of the users' sight. As they see errors or unusual behavior, one of the evaluators marks the video and records a brief remark. When the test is finished evaluators can use the annotated recording to calculate performance times so they can compared users' performance on different prototypes. The data stream iron: the interaction log is used in a similar way to calculate performance times. Typically this data is further analyzed using simple statistics such as means, standard deviations, T-tests, etc. Categorized data may also be quantified and analyzed statistically, as we have said.

Feeding the findings back into design

The results from an evaluation can be reported to the design team in several ways, as we have indicated. Clearly written reports with an overview at the beginning and detailed content list make for easy reading and a good reference document. Including anecdotes, quotations, pictures, and video clips helps to bring the study to life, stimulate interest, and make the written description more meaningful. Some teams like quantitative data, but its

© Copyright Virtual University of Pakistan

value depends on the type of study and its goals. Verbal presentations that include video clips can also be very powerful. Often both qualitative and quantitative data analysis are useful because they provide alternative perspectives.

© Copyright Virtual University of Pakistan

Lecture 41.

Asking Users

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Discuss when it is appropriate to use different types of interviews and questionnaires.
- Teach you the basics of questionnaire design.
- Describe how to do interviews, heuristic evaluation, and walkthroughs.
- Describe how to collect, analyze, and present data collected by the techniques mentioned above.
- Enable you to discuss the strengths and limitations of the techniques and select appropriate ones for your own use.

41.1 Introduction

In the last lecture we looked at observing users. Another way of finding out what users do, what they want to do like, or don't like is to ask them. Interviews and questionnaires are well-established techniques in social science research, market research, and human-computer interaction. They are used in "quick and dirty" evaluation, in usability testing, and in field studies to ask about *facts*, *behavior*, *beliefs*, and *attitudes*. Interviews and questionnaires can be structured or flexible and more like a discussion, as in field studies. Often interviews and observation go together in field studies, but in this lecture we focus specifically on interviewing techniques.

The first part of this lecture discusses interviews and questionnaires. As with observation, these techniques can be used in the requirements activity, but in this lecture we focus on their use in evaluation. Another way of finding out how well a system is designed is by asking experts for then opinions. In the second part of the lecture, we look at the techniques of heuristic evaluation and cognitive walkthrough. These methods involve predicting how usable interfaces are (or are not).

41.2 Asking users: interviews

Interviews can be thought of as a "conversation with a purpose" (Kahn and Cannell, 1957). How like an ordinary conversation the interview is depends on the " questions to be answered and the type of interview method used. There are four main types of interviews: *open-ended or unstructured, structured, semi-structured,* and *group* interviews (Fontana and Frey, 1994). The first three types are named according to how much control the interviewer imposes on the conversation by following a *predetermined set of questions.* The fourth involves a small group guided by an interviewer who facilitates discussion of a specified set of topics.

The most appropriate approach to interviewing depends on the evaluation goals, the questions to be addressed, and the paradigm adopted. For example, it the goal is to gain first impressions about how users react to a new design idea, such as an interactive sign, then an informal, open-ended interview is often the best approach. But if the goal is to get

© Copyright Virtual University of Pakistan

feedback about a particular design feature, such as the layout of a new web browser, then a structured interview or questionnaire is often better. This is because the goals and questions are more specific in the latter case.

Developing questions and planning an interview

When developing interview questions, plan to keep them short, straightforward and avoid asking too many. Here are some guidelines (Robson, 1993):

- Avoid long questions because they are difficult to remember.
- Avoid compound sentences by splitting them into two separate questions. For example, instead of, "How do you like this cell phone compared with previous ones that you have owned?" Say, "How do you like this cell phone? Have you owned other cell phones? If so, "How did you like it?" This is easier for the interviewee and easier for the interviewer to record.
- Avoid using jargon and language that the interviewee may not understand but would be too embarrassed to admit.
- Avoid leading questions such as, "Why do you like this style of interaction?" It used on its own, this question assumes that the person did like it.
- Be alert to unconscious biases. Be sensitive to your own biases and strive for neutrality in your questions.

Asking colleagues to review the questions and running a pilot study will help to identify problems in advance and gain practice in interviewing.
When planning an interview, think about interviewees who may be reticent to answer questions or who are in a hurry. They are doing *you a* favor, so try to make it as pleasant for them as possible and try to make the interviewee feel comfortable. Including the following steps will help you to achieve this (Robson, 1993):

1. An *Introduction* in which the interviewer introduces himself and explains why he is doing the interview, reassures interviewees about the ethical issues, and asks if they mind being recorded, if appropriate. This should be exactly the same for each interviewee.
2. A *warmup* session where easy, non-threatening questions come first. These may include questions about demographic information, such as "Where do you live?"
3. A *main* session in which the questions are presented in a logical sequence, with the more difficult ones at the end.
4. A *cool-off period* consisting of a few easy questions (to defuse tension if it has arisen).
5. A *closing* session in which the interviewer thanks the interviewee and switches off the recorder or puts her notebook away, signaling that the interview has ended.

The golden rule is to be professional. Here is some further advice about conducting interviews (Robson. 1993):

- Dress in a similar way to the interviewees if possible. If in doubt, dress neatly and avoid standing out.
- Prepare an informed consent form and ask the interviewee to sign it.
- If you are recording the interview, which is advisable, make sure your equipment works in advance and you know how to use it.

© Copyright Virtual University of Pakistan

- Record answers exactly: do not make cosmetic adjustments, correct, or change answers in any way.

Unstructured interviews

Open-ended or unstructured interviews are at one end of a spectrum of how much control the interviewer has on the process. They are more like conversations that focus on a particular topic and may often go into considerable depth. Questions posed by the interviewer are *open*, meaning that the format and content of answers is not predetermined. The interviewee is free to answer as fully or as briefly as she wishes. Both interviewer and interviewee can steer the interview. Thus one of the skills necessary for this type of interviewing is to make sure that answers to relevant questions are obtained. It is therefore advisable to be organized and have a plan of the main things to be covered. Going in without an agenda to accomplish a goal is *not* advisable, and should not to be confused with being open to new information and ideas.

A benefit of unstructured interviews is that they generate rich data. Interviewees often mention things that the interviewer may not have considered and can be further explored. But this benefit often comes at a cost. A lot of unstructured data is generated, which can be very time-consuming and difficult to analyze. It is also impossible to replicate the process, since each interview takes on its own format. Typically in evaluation, there is no attempt to analyze these interviews in detail. Instead, the evaluator makes notes or records the session and then goes back later to note the main issues of interest.

The main points to remember when conducting an unstructured interview are:

- Make sure you have an interview agenda that supports the study goals and questions (identified through the DECIDE framework).
- Be prepared to follow new lines of enquiry that contribute to your agenda.
- Pay attention to ethical issues, particularly the need to get informed consent.
- Work on gaining acceptance and putting the interviewees at ease. For example, dress as they do and take the time to learn about their world.
- Respond with sympathy if appropriate, but be careful not to put ideas into the heads of respondents.
- Always indicate to the interviewee the beginning and end of the interview session.
- Start to order and analyze your data as soon as possible after the interview

Structured interviews

Structured interviews pose predetermined questions similar to those in a questionnaire. Structured interviews are useful when the study's goals arc clearly understood and specific questions can he identified. To work best, the questions need to he short and clearly worded. Responses may involve selecting from a set of options that are read aloud or presented on paper. The questions should be refined by asking another evaluator to review them and by running a small pilot study. Typically the questions are closed, which means that they require a precise answer. The same questions are used with each participant so the study is standardized.

Semi-structured interviews

Semi-structured interviews combine features of structured and unstructured inter views and use both closed and open questions. For consistency the interviewer has a basic script for guidance, so that the same topics arc covered with each interviewee.

© Copyright Virtual University of Pakistan

The interviewer starts with preplanned questions and then probes the interviewee to say more until no new relevant information is forthcoming. For example:

> Which websites do you visit most frequently? <Answer> Why? <Answer mentions several but stresses that prefers hottestmusic.com> And why do you like it? <Answer> Tell me more about x? <Silence, followed by an answer> Anything else? <Answer>Thanks. Are there any other reasons that you haven't mentioned?

It is important not to preempt an answer by phrasing a question to suggest that a particular answer is expected. For example. "You seemed to like this use of color…" assumes that this is the case and will probably encourage the interviewee to answer that this is true so as not to offend the interviewer. Children are particularly prone to behave in this way. The body language of the interviewer, for example, whether she is smiling, scowling, looking disapproving, etc., can have a strong influence.
Also the interviewer needs to accommodate silence and not to move on too quickly. Give the person time to speak. Probes are a device for getting more information, especially neutral probes such as, "Do you want to tell me anything else" You may also prompt the person to help her along. For example, if the interviews is talking about a computer interface hut has forgotten the name of a key menu item, you might want to remind her so that the interview can proceed productively However, semi-structured interviews are intended to be broadly replicable. So probing and prompting should aim to help the interview along without introducing bias

Group interviews

One form of group interview is the focus group that is frequently used in marketing, political campaigning, and social sciences research. Normally three to 10 people are involved. Participants are selected to provide a representative sample of typical users; they normally share certain characteristics. For example, in an evaluation of a university website, a group of administrators, faculty, and students may be called to form three separate focus groups because they use the web for different purposes.
The benefit of a focus group is that it allows diverse or sensitive issues to be raised that would otherwise be missed. The method assumes that individuals develop opinions within a social context by talking with others. Often questions posed to focus groups seem deceptively simple but the idea is to enable people to put forward their own opinions in a supportive environment. A preset agenda is developed to guide the discussion but there is sufficient flexibility for a facilitator to follow unanticipated issues as they are raised. The facilitator guides and prompts discussion and skillfully encourages quiet people to participate and stops verbose ones from dominating the discussion. The discussion is usually recorded for later analysis in which participants may be invited to explain their comments more fully.

Focus groups appear to have high validity because the method is readily understood and findings appear believable (Marshall and Rossman, 1999). Focus groups are also attractive because they are low-cost, provide quick results, and can easily be scaled to gather more data. Disadvantages are that the facilitator needs to be skillful so that time is not wasted on irrelevant issues. It can also be difficult to get people together in a suitable location. Getting time with any interviewees can be difficult, but the problem is compounded with focus groups because of the number of people involved. For example, in a study to evaluate a university website the evaluators did not expect that getting

© Copyright Virtual University of Pakistan

participants would be a problem. However, the study was scheduled near the end of a semester when students had to hand in their work, so strong incentives were needed to entice the students to participate in the study. It took an increase in the participation fee and a good lunch to convince students to participate.

Other sources of interview-like feedback

Telephone interviews are a good way of interviewing people with whom you cannot meet. You cannot see body language, but apart from this telephone interviews have much in common with face-to-face interviews.

Online interviews, using either asynchronous communication as in email or synchronous communication as in chats, can also be used. For interviews that involve sensitive issues, answering questions anonymously may be preferable to meeting face to face. If, however, face-to-face meetings are desirable but impossible because of geographical distance, video-conferencing systems can be used. Feedback about a product can also be obtained from customer help lines, consumer groups, and online customer communities that provide help and support.

At various stages of design, it is useful to get quick feedback from a few users. These short interviews are often more like conversations in which users are asked their opinions. Retrospective interviews can be done when doing field studies to check with participants that the interviewer has correctly understood what was happening.

Data analysis and interpretation

Analysis of unstructured interviews can be time-consuming, though their contents can be rich. Typically each interview question is examined in depth in a similar way to observation data. A coding form may he developed, which may he predetermined or may he developed during data collection as evaluators are exposed to the range of issues and learn about their relative importance Alternatively, comments may he clustered along themes and anonymous quotes used to illustrate points of interest. Tools such a NUDIST and Ethnography can be useful for qualitative analyses. Which type of analysis is done depends on the goals of the study, as does whether the whole interview is transcribed, only part of it, or none of it. Data from structured interviews is usually analyzed quantitatively as in questionnaires, which we discuss next.

41.3 Asking users: questionnaires

Questionnaires are a well-established technique for collecting demographic data and users' opinions. They are similar to interviews and can have *closed* or *open* questions. Effort and skill are needed to ensure that questions are clearly worded and the data collected can be analyzed efficiently. Questionnaires can be used on their own or in conjunction with other methods to clarify or deepen understanding.

The questions asked in a questionnaire, and those used in a structured interview are similar, so how do you know when to use which technique? One advantage of questionnaires is that they can be distributed to a large number of people. Used in this way, they provide evidence of wide general opinion. On the other hand, structured interviews are easy and quick to conduct in situations in which people will not stop to complete a questionnaire.

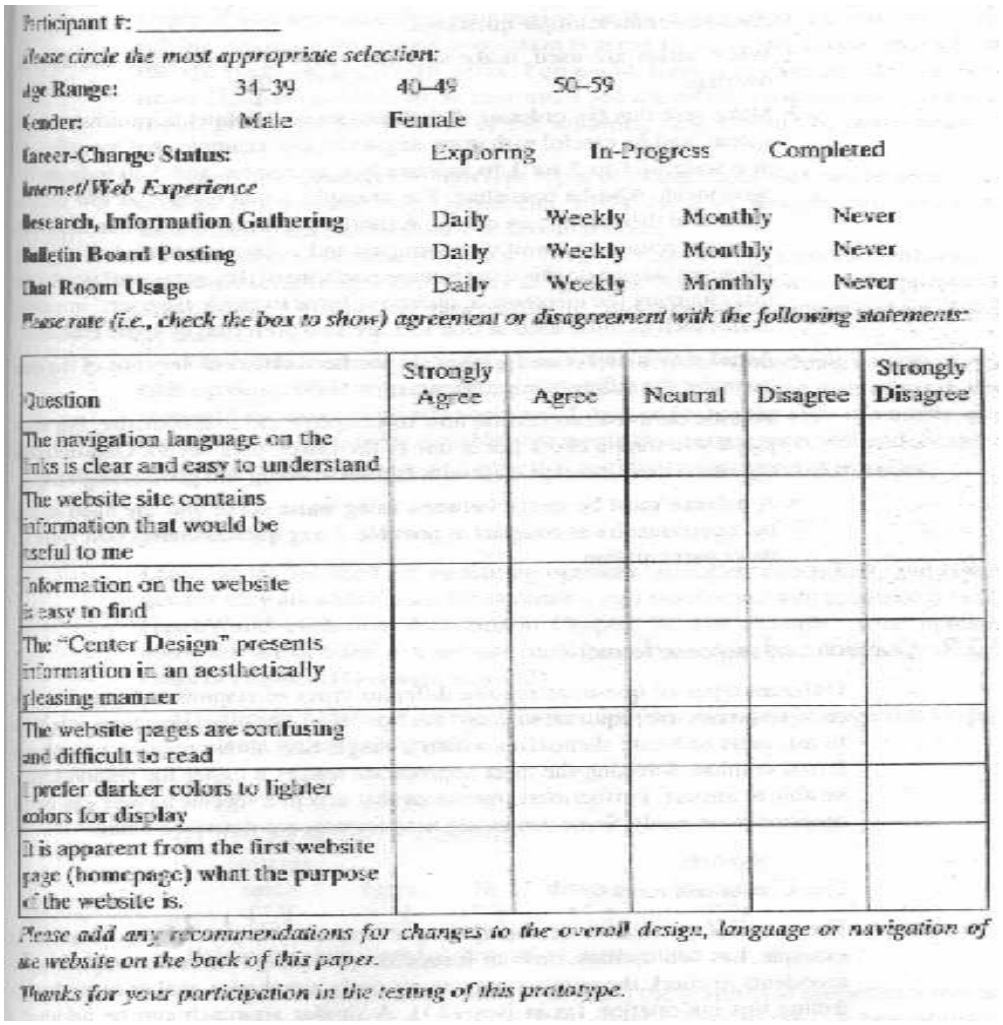
Designing questionnaires

Many questionnaires start by asking for basic demographic information (e.g.. gender. age) and details of user experience (e.g., the time or number of years spent using

© Copyright Virtual University of Pakistan

computers, level of expertise, etc.). This background information is useful in finding out the range within the sample group. For instance, a group of people who are using the web for the first time are likely to express different opinions to another group with five years of web experience. From knowing the sample range, a designer might develop two different versions or veer towards the needs of one of the groups more because it represents the target audience.

Following the general questions, specific questions that contribute to the evaluation goal are asked. If the questionnaire is long, the questions may be subdivided into related topics to make it easier and more logical to complete. Figure below contains an excerpt from a paper questionnaire designed to evaluate users" satisfaction with some specific features of a prototype website for career changers aged *34-59* years.



Participant #: _____

Please circle the most appropriate selection:

Age Range: 34–39 40–49 50–59

Gender: Male Female

Career-Change Status: Exploring In-Progress Completed

Internet/Web Experience

Research, Information Gathering	Daily	Weekly	Monthly	Never
Bulletin Board Posting	Daily	Weekly	Monthly	Never
Chat Room Usage	Daily	Weekly	Monthly	Never

Please rate (i.e., check the box to show) agreement or disagreement with the following statements:

Question	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
The navigation language on the links is clear and easy to understand					
The website site contains information that would be useful to me					
Information on the website is easy to find					
The "Center Design" presents information in an aesthetically pleasing manner					
The website pages are confusing and difficult to read					
I prefer darker colors to lighter colors for display					
It is apparent from the first website page (homepage) what the purpose of the website is.					

Please add any recommendations for changes to the overall design, language or navigation of the website on the back of this paper.

Thanks for your participation in the testing of this prototype.

The following is a checklist of general advice for designing a questionnaire:

- Make questions clear and specific.
- When possible, ask closed questions and offer a range of answers.
- Consider including a "no-opinion" option for questions that seek opinions.
- Think about the ordering of questions. The impact of a question can he influenced by question order. General questions should precede specific ones.
- Avoid complex multiple questions.

© Copyright Virtual University of Pakistan

- When scales are used, make sure the range is appropriate and does not overlap.
- Make sure that the ordering of scales (discussed below) is intuitive and consistent, and be careful with using negatives. For example, it is more intuitive in a scale of 1 to 5 for 1 to indicate low agreement and 5 to indicate high agreement. Also be consistent. For example, avoid using 1 as low on some scales and then as high on others. A subtler problem occurs when most questions are phrased as positive statements and a few are phrased as negatives. However, advice on this issue is more controversial as some evaluators argue that changing the direction of questions helps to check the users' intentions.
- Avoid jargon and consider whether you need different versions of the questionnaire for different populations.
- Provide clear instructions on how to complete the questionnaire. For example, if you want a check put in one of the boxes, then say so. Questionnaires can make their message clear with careful wording and good typography.
- A balance must be struck between using white space and the need to keep the questionnaire as compact as possible. Long questionnaires cost more and deter participation.

Question and response format

Different types of questions require different types of responses. Sometimes discrete responses arc required, such as "Yes" or "No." For other questions it is better to ask users to locate themselves within a range. Still others require a single preferred opinion. Selecting the most appropriate makes it easier for respondents to be able to answer. Furthermore, questions that accept a specific answer can be categorized more easily. Some commonly used formats are described below.

Check boxes and ranges

The range of answers to demographic questionnaires is predictable. Gender, for example, has two options, male or female, so providing two boxes and asking respondents to check the appropriate one, or circle a response, makes sense for collecting this information. A similar approach can be adopted if details of age are needed. But since some people do not like to give their exact age many questionnaires ask respondents to specify their age as a range. A common design error arises when the ranges overlap. For example, specifying two ranges as 15-20, 20-25 will cause confusion: which box do people who are 20 years old check? Making the ranges 14-19, 20-24 avoids this problem.
A frequently asked question about ranges is whether the interval must be equal in all cases. The answer is that it depends on what you want to know. For example, if you want to collect information for the design of an e-commerce site to sell life insurance, the target population is going to be mostly people with jobs in the age range of, say, 21-65 years. You could, therefore, have just three ranges: under 21, 21-65 and over 65. In contrast, if you are interested in looking at ten-year cohort groups for people over 21 the following ranges would he best: under 21, 22-31, 32-41, etc.

Administering questionnaires

Two important issues when using questionnaires are reaching a representative sample of participants and ensuring a reasonable response rate. For large surveys, potential respondents need to be selected using a sampling technique. However, interaction designers tend to use small numbers of participants, often fewer than

© Copyright Virtual University of Pakistan

twenty users. One hundred percent completion rates often are achieved with these small samples, but with larger, more remote populations, ensuring that surveys are returned is a well-known problem. Forty percent return is generally acceptable for many surveys but much lower rates are common.
Some ways of encouraging a good response include:

- Ensuring the questionnaire is well designed so that participants do not get annoyed and give up.
- Providing a short overview section and telling respondents to complete just the short version if they do not have time to complete the whole thing. This ensures that you get something useful returned.
- Including a stamped, self-addressed envelope for its return.
- Explaining why you need the questionnaire to be completed and assuring anonymity.
- Contacting respondents through a follow-up letter, phone call or email.
- Offering incentives such as payments.

Online questionnaires

Online questionnaires are becoming increasingly common because they are effective for reaching large numbers of people quickly and easily. There are two types: email and web-based. The main advantage of email is that you can target specific users. However, email questionnaires are usually limited to text, whereas web-based questionnaires are more flexible and can include check boxes, pull-down and pop-up menus, help screens, and graphics, web-based questionnaires can also provide immediate data validation and can enforce rules such as select only one response, or certain types of answers such as numerical, which cannot be done in email or with paper. Other advantages of online questionnaires include (Lazar and Preece, 1999):

- Responses are usually received quickly.
- Copying and postage costs are lower than for paper surveys or often nonexistent.
- Data can be transferred immediately into a database for analysis.
- The time required for data analysis is reduced.
- Errors in questionnaire design can be corrected easily (though it is better to avoid them in the first place).

A big problem with web-based questionnaires is obtaining a random sample of respondents. Few other disadvantages have been reported with online questionnaires, but there is some evidence suggesting that response rates may be lower online than with paper questionnaires (Witmer et al., 1999).

Heuristic evaluation

Heuristic evaluation is an informal usability inspection technique developed by Jakob Nielsen and his colleagues (Nielsen, 1994a) in which experts, guided by a set of usability principles known as *heuristics,* evaluate whether user-interface elements, such as dialog boxes, menus, navigation structure, online help, etc., conform to the principles. These heuristics closely resemble the high-level design principles and guidelines e.g., making designs consistent, reducing memory load, and using terms that users understand. When used in evaluation, they are called heuristics. The original set of heuristics was derived empirically from an analysis of 249 usability problems

© Copyright Virtual University of Pakistan

(Nielsen, 1994b). We list the latest here, this time expanding them to include some of the questions addressed when doing evaluation:

- *Visibility of system status*
  - Are users kept informed about what is going on?
  - Is appropriate feedback provided within reasonable time about a user's action?
- *Match between system and the real world*
  - Is the language used at the interface simple?
  - Are the words, phrases and concepts used familiar to the user?
- *User control and freedom*
  - Are there ways of allowing users to easily escape from places they unexpectedly find themselves in?
- *Consistency and standards*
  - Are the ways of performing similar actions consistent?
- *Help users recognize, diagnose, and recover from errors*
  - Are error messages helpful?
  - Do they use plain language to describe the nature of the problem and suggest a way of solving it?
- *Error prevention*
  - Is it easy to make errors?
  - If so where and why?
- *Recognition rather than recall*
  - Are objects, actions and options always visible?
- *Flexibility and efficiency of use*
  - Have accelerators (i.e., shortcuts) been provided that allow more experienced users to carry out tasks more quickly?
- *Aesthetic and minimalist design*
  - Is any unnecessary and irrelevant information provided?
- *Help and documentation*
  - Is help information provided that can be easily searched and easily followed'.'

However, some of these core heuristics are too general for evaluating new products coming onto the market and there is a strong need for heuristics that are more closely tailored to specific products. For example, Nielsen (1999) suggests that the following heuristics are more useful for evaluating commercial websites and makes them memorable by introducing the acronym HOME RUN:

- High-quality content
- Often updated
- Minimal download time
- Ease of use
- Relevant to users' needs
- Unique to the online medium
- Net-centric corporate culture

Different sets of heuristics for evaluating toys, WAP devices, online communities, wearable computers, and other devices are needed, so evaluators must develop their own by tailoring Nielsen's heuristics and by referring to design guidelines, market research, and requirements documents. Exactly which heuristics are the best and how many are needed are debatable and depend on the product.

© Copyright Virtual University of Pakistan

Using a set of heuristics, expert evaluators work with the product role-playing typical users and noting the problems they encounter. Although other numbers of experts can be used, empirical evidence suggests that five evaluators usually identify around 75% of the total usability problems.

41.4 Asking experts: walkthroughs

Walkthroughs are an alternative approach to heuristic evaluation for predicting users' problems without doing user testing. As the name suggests, they involve walking through a task with the system and noting problematic usability features. Most walkthrough techniques do not involve users. Others, such as pluralistic walkthroughs, involve a team that includes users, developers, and usability specialists.
In this section we consider cognitive and pluralistic walkthroughs. Both were originally developed for desktop systems but can be applied to web-based systems, handheld devices, and products such as VCRs,

Cognitive walkthroughs

"Cognitive walkthroughs involve simulating a user's problem-solving process at each step in the human-computer dialog, checking to see if the user's goals and memory for actions can be assumed to lead to the next correct action." (Nielsen and Mack, 1994, p. 6). The defining feature is that they focus on evaluating designs for ease of learning—a focus that is motivated by observations that users learn by exploration (Wharton et al., 1994). The steps involved in cognitive walkthroughs are:

1. The characteristics of typical users are identified and documented and sample tasks are developed that focus on the aspects of the design to be evaluated. A description or prototype of the interface to be developed is also produced, along with a clear sequence of the actions needed for the users to complete the task.
2. A designer and one or more expert evaluators then come together to do the analysis.
3. The evaluators walk through the action sequences for each task, placing H within the context of a typical scenario, and as they do this they try to answer the following questions:

    - Will the correct action be sufficiently evident to the user? (Will the user know what to do to achieve the task?)
    - Will the user notice that the correct action is available? (Can users see the button or menu item that they should use for the next action? Is it apparent when it is needed?)
    - Will the user associate and interpret the response from the action correctly? (Will users know from the feedback that they have made a correct or incorrect choice of action?)

    In other words: will users know what to do, see how to do it, and understand from feedback whether the action was correct or not?

4. As the walkthrough is being done, a record of critical information is compiled in which:

    - The assumptions about what would cause problems and why are recorded. This involves explaining why users would face difficulties.

© Copyright Virtual University of Pakistan

- Notes about side issues and design changes are made.
- A summary of the results is compiled.

5. The design is then revised to fix the problems presented.

It is important to document the cognitive walkthrough, keeping account of what works and what doesn't. A standardized feedback form can be used in which answers are recorded to the three bulleted questions in step (3) above. The form can also record the details outlined in points 1-4 as well as the date of the evaluation. Negative answers to any of the questions are carefully documented on a separate form, along with details of the system, its version number, the date of the evaluation, and the evaluators' names. It is also useful to document the severity of the problems, for example, how likely a problem is to occur and how serious it will be for users.
The strengths of this technique are that it focuses on users" problems in detail, yet users do not need to be present, nor is a working prototype necessary. However, it is very time-consuming and laborious to do. Furthermore the technique has a narrow focus that can be useful for certain types of system but not others.

Pluralistic walkthroughs

"Pluralistic walkthroughs are another type of walkthrough in which users, developers and usability experts work together to step through a [task] scenario, discussing usability issues associated with dialog elements involved in the scenario steps" (Nielsen and Mack, 1994. p. 5). Each group of experts is asked to assume the role of typical users. The walkthroughs are then done by following a sequence of steps (Bias, 1994):

1. Scenarios are developed in the form of a series of hard-copy screens representing a single path through the interface. Often just two or a few screens are developed.
2. The scenarios are presented to the panel of evaluators and the panelists are asked to write down the sequence of actions they would take to move from one screen to another. They do this individually without conferring with one another.
3. When everyone has written down their actions, the panelists discuss the actions that they suggested for that round of the review. Usually, the representative users go first so that they are not influenced by the other panel members and are not deterred from speaking. Then the usability experts present their findings, and finally the developers offer their comments.
4. Then the panel moves on to the next round of screens. This process continues until all the scenarios have been evaluated.

The benefits of pluralistic walkthroughs include a strong focus on users' tasks. Performance data is produced and many designers like the apparent clarity of working with quantitative data. The approach also lends itself well to participatory design practices by involving a multidisciplinary team in which users play a key role. Limitations include having to get all the experts together at once and then proceed at the rate of the slowest. Furthermore, only a limited number of scenarios, and hence paths through the interface, can usually be explored because of time constraints.

© Copyright Virtual University of Pakistan

Lecture 42.

Communicating Users

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:
Discuss how to eliminate error messages
Learn how to eliminate notifiers and confirmatory messages

42.1 Eliminating Errors

Bulletin dialog boxes are used for error messages, notifiers, and confirmations, three of the most abused components of modern GUI design. With proper design, these dialogs can all but be eliminated. In this lecture, we'll explore how and why.

Errors Are Abused

There is probably no more abused idiom in the GUI world than the error dialog. The proposal that a program doesn't have the right — even the duty — to reject the user's input is so heretical that many practitioners dismiss it summarily. Yet, if we examine this assertion rationally and from the user's — rather than the programmer's — point of view, it is not only possible, but quite reasonable.
Users never want error messages. Users want to avoid the consequences of making errors, which is very different from saying that they want error messages. It's like saying that people want to abstain from skiing when what they really want to do is avoid breaking their legs. Usability guru Donald Norman (1989) points out that users frequently blame themselves for errors in product design. Just because you aren't getting complaints from your users doesn't mean that they are happy getting error messages.

Why We Have So Many Error Messages

The first computers were undersized, underpowered, and expensive, and didn't lend themselves easily to software sensitivity. The operators of these machines were white-lab-coated scientists who were sympathetic to the needs of the CPU and weren't offended when handed an error message. They knew how hard the computer was working. They didn't mind getting a core dump, a bomb, an "Abort, Retry, Fail?" or the infamous "FU" message (File Unavailable). This is how the tradition of software treating people like CPUs began. Ever since the early days of computing, programmers have accepted that the proper way for software to interact with humans was to demand input and to complain when the human failed to achieve the same perfection level as the CPU.
Examples of this approach exist wherever software demands that the user do things its way instead of the software adapting to the needs of the human. Nowhere is it more prevalent, though, than in the omnipresence of error messages.

What's Wrong with Error Messages

Error messages, as blocking modal bulletins must stop the proceedings with a modal dialog box. Most user interface designers — being programmers — imagine that their

© Copyright Virtual University of Pakistan

error message boxes are alerting the user to serious problems. This is a widespread misconception. Most error message boxes are informing the user of the inability of the program to work flexibly. Most error message boxes seem to the user like an admission of real stupidity on the program's part. In other words, to most users, error message boxes are seen not just as the program stopping the proceedings but, in clear violation of the axiom: Don't stop the proceedings with idiocy. We can significantly improve the quality of our interfaces by eliminating error message boxes.

People hate error messages

Humans have emotions and feelings: Computers don't. When one chunk of code rejects the input of another, the sending code doesn't care; it doesn't scowl, get hurt, or seek counseling. Humans, on the other hand, get angry when they are flatly told they are idiots. When users see an error message box, it is as if another person has told them that they are stupid. Users hate this. Despite the inevitable user reaction, most programmers just shrug their shoulders and put error message boxes in anyway. They don't know how else to create reliable software.

Many programmers and user interface designer’s labor under the misconception that people either like or need to be told when they are wrong. This assumption is false in several ways. The assumption that people like to know when they are wrong ignores human nature. Many people become very upset when they are informed of their mistakes and would rather not know that they did something wrong. Many people don't like to hear that they are wrong from anybody but themselves. Others are only willing to hear it from a spouse or close friend. Very few wish to hear about it from a machine. You may call it denial, but it is true, and users will blame the messenger before they blame themselves.

The assumption that users need to know when they are wrong is similarly false. How important is it for you to know that you requested an invalid type size? Most programs can make a reasonable substitution.

We consider it very impolite to tell people when they have committed some social faux pas. Telling someone they have a bit of lettuce sticking to their teeth or that their fly is open is equally embarrassing for both parties. Sensitive people look for ways to bring the problem to the attention of the victim without letting others notice. Yet programmers assume that a big, bold box in the middle of the screen that stops all the action and emits a bold "beep" is the appropriate way to behave.

Whose mistake is it, anyway?

Conventional wisdom says that error messages tell the user when he has made some mistake. Actually, most error bulletins report to the user when the program gets confused. Users make far fewer substantive mistakes than imagined. Typical "errors" consist of the user inadvertently entering an out-of-bounds number, or entering a space where the computer doesn't allow it. When the user enters something unintelligible by the computer's standards, whose fault is it? Is it the user's fault for not knowing how to use the program properly, or is it the fault of the program for not making the choices and effects clearer?

Information that is entered in an unfamiliar sequence is usually considered an error by software, but people don't have this difficulty with unfamiliar sequences. Humans know how to wait, to bide their time until the story is complete. Software usually jumps to the erroneous conclusion that out-of-sequence input means wrong input and issues the evil error message box.

© Copyright Virtual University of Pakistan

When, for example, the user creates an invoice for an invalid customer number, most programs reject the entry. They stop the proceedings with the idiocy that the user must make the customer number valid right now. Alternatively, the program could accept the transaction with the expectation that a valid customer number will eventually be entered. It could, for example, make a special notation to itself indicating what it lacks. The program then watches to make sure the user enters the necessary information to make that customer number valid before the end of the session, or even the end of the month book closing. This is the way most humans work. They don't usually enter "bad" codes. Rather, they enter codes in a sequence that the software isn't prepared to accept.

If the human forgets to fully explain things to the computer, it can after some reasonable delay, provide more insistent signals to the user. At day's or week's end the program can move irreconcilable transactions into a suspense account. The program doesn't have to bring the proceedings to a halt with an error message. After all, the program will remember the transactions so they can be tracked down and fixed. This is the way it worked in manual systems, so why can't computerized systems do at least this much? Why stop the entire process just because something is missing? As long as the user remains well informed throughout that some accounts still need tidying, there shouldn't be a problem. The trick is to inform without stopping the proceedings.

If the program were a human assistant and it staged a sit-down strike in the middle of the accounting department because we handed it an incomplete form, we'd be pretty upset. If we were the bosses, we'd consider finding a replacement for this anal-retentive, petty, sanctimonious clerk. Just take the form, we'd say, and figure out the missing information. The experts have used Rolodex programs that demand you enter an area code with a phone number even though the person's address has already been entered. It doesn't take a lot of intelligence to make a reasonable guess at the area code. If you enter a new name with an address in Menlo Park, the program can reliably assume that their area code is 650 by looking at the other 25 people in your database who also live in Menlo Park and have 650 as their area code. Sure, if you enter a new address for, say, Boise, Idaho, the program might be stumped. But how tough is it to access a directory on the Web, or even keep a list of the 1,000 biggest cities in America along with their area codes?

Programmers may now protest: "The program might be wrong. It can't be sure. Some cities have more than one area code. It can't make that assumption without approval of the user!" Not so.

If we asked a human assistant to enter a client's phone contact information into our Rolodex, and neglected to mention the area code, he would accept it anyway, expecting that the area code would arrive before its absence was critical. Alternatively, he could look the address up in a directory. Let's say that the client is in Los Angeles so the directory is ambiguous: The area code could be either 213 or 310. If our human assistant rushed into the office in a panic shouting "Stop what you're doing! This client's area code is ambiguous!" we'd be sorely tempted to fire him and hire somebody with a greater-than-room-temperature IQ. Why should software be any different? A human might write 213/310? into the area code field in this case. The next time

we call that client, we'll have to determine which area code is correct, but in the meantime, life can go on.

Again, squeals of protest: "But the area code field is only big enough for three digits! I can't fit 213/310? into it!" Gee, that's too bad. You mean that rendering the user interface of your program in terms of the underlying implementation model — a rigidly fixed field width — forces you to reject natural human behavior in favor of obnoxious, computer-like inflexibility supplemented with demeaning error messages? Not to put too fine a

© Copyright Virtual University of Pakistan

point on this, but error message boxes come from a failure of the program to behave reasonably, not from any failure of the user.

This example illustrates another important observation about user interface design. It is not only skin deep. Problems that aren't solved in the design are pushed through the system until they fall into the lap of the user. There are a variety of ways to handle the exceptional situations that arise in interaction with software — and a creative designer or programmer can probably think of a half-dozen or so off the top of her head — but most programmers just don't try. They are compromised by their schedule and their preferences, so they tend to envision the world in the terms of perfect CPU behavior rather than in the terms of imperfect human behavior.

Error messages don't work

There is a final irony to error messages: They don't prevent the user from making errors. We imagine that the user is staying out of trouble because our trusty error messages keep them straight, but this is a delusion. What error messages really do is prevent the program from getting into trouble. In most software, the error messages stand like sentries where the program is most sensitive, not where the user is most vulnerable, setting into concrete the idea that the program is more important than the user. Users get into plenty of trouble with our software, regardless of the quantity or quality of the error messages in it. All an error message can do is keep me from entering letters in a numeric field — it does nothing to protect me from entering the wrong numbers — which is a much more difficult design task.

Eliminating Error Messages

We can't eliminate error messages by simply discarding the code that shows the actual error message dialog box and letting the program crash if a problem arises. Instead, we need to rewrite the programs so they are no longer susceptible to the problem. We must replace the error-message with a kinder, gentler, more robust software that prevents error conditions from arising, rather than having the program merely complain when things aren't going precisely the way it wants. Like vaccinating it against a disease, we make the program immune to the problem, and then we can toss the message that reports it. To eliminate the error message, we must first eliminate the possibility of the user making the error. Instead of assuming error messages are normal, we need to think of them as abnormal solutions to rare problems — as surgery instead of aspirin. We need to treat them as an idiom of last resort.

Every good programmer knows that if module A hands invalid data to module B, module B should clearly and immediately reject the input with a suitable error indicator. Not doing this would be a great failure in the design of the interface between the modules. But human users are not modules of code. Not only should software not reject the input with an

error message, but the software designer must also reevaluate the entire concept of what "invalid data" is. When it comes from a human, the software must assume that the input is correct, simply because the human is more important than the code. Instead of software rejecting input, it must work harder to understand and reconcile confusing input. The program may understand the state of things inside the computer, but only the user understands the state of things in the real world. Ultimately, the real world is more relevant and important than what the computer thinks.

© Copyright Virtual University of Pakistan

Making errors impossible

Making it impossible for the user to make errors is the best way to eliminate error messages. By using bounded gizmos for all data entry, users are prevented from ever being able to enter bad numbers. Instead of forcing a user to key in his selection, present him with a list of possible selections from which to choose. Instead of making the user type in a state code, for example, let him choose from a list of valid state codes or even from a picture of a map. In other words, make it impossible for the user to enter a bad state.

Another excellent way to eliminate error messages is to make the program smart enough that it no longer needs to make unnecessary demands. Many error messages say things like "Invalid input. User must type xxxx." Why can't the program, if it knows what the user must type, just enter xxxx by itself and save the user the tongue-lashing? Instead of demanding that the user find a file on a disk, introducing the chance that the user will select the wrong file, have the program remember which files it has accessed in the past and allow a selection from that list. Another example is designing a system that gets the date from the internal clock instead of asking for input from the user.

Undoubtedly, all these solutions will cause more work for programmers. However, it is the programmer's job to satisfy the user and not vice versa. If the programmer thinks of the user as just another input device, it is easy to forget the proper pecking order in the world of software design.

Users of computers aren't sympathetic to the difficulties faced by programmers. They don't see the technical rationale behind an error message box. All they see is the unwillingness of the program to deal with things in a human way.

One of the problems with error messages is that they are usually post facto reports of failure. They say, "Bad things just happened, and all you can do is acknowledge the catastrophe." Such reports are not helpful. And these dialog boxes always come with an OK button, requiring the user to be an accessory to the crime. These error message boxes are reminiscent of the scene in old war movies where an ill-fated soldier steps on a landmine while advancing across the rice paddy. He and his buddies clearly hear the click of the mine's triggering mechanism and the realization comes over the soldier that although he's safe now, as soon as he removes his foot from the mine, it will explode, taking some large and useful part of his body with it. Users get this feeling when they see most error message boxes, and they wish they were thousands of miles away, back in the real world.

42.2 Positive feedback

One of the reasons why software is so hard to learn is that it so rarely gives positive feedback. People learn better from positive feedback than they do from negative feedback. People want to use their software correctly and effectively, and they are motivated to learn how to make the software work for them. They don't need to be slapped on the wrist when they fail. They do need to be rewarded, or at least acknowledged, when they succeed. They will feel better about themselves if they get approval, and that good feeling will be reflected back to the product.

Advocates of negative feedback can cite numerous examples of its effectiveness in guiding people's behavior. This evidence is true, but almost universally, the context of effective punitive feedback is getting people to refrain from doing things they want to do but shouldn't: Things like not driving over 55 mph, not cheating on their spouses, and not

© Copyright Virtual University of Pakistan

fudging their income taxes. But when it comes to helping people do what they want to do, positive feedback is best. Imagine a hired ski instructor who yells at you, or a restaurant host who loudly announces to other patrons that your credit card was rejected.

Keep in mind that we are talking about the drawbacks of negative feedback from a computer. Negative feedback by another person, although unpleasant, can be justified in certain circumstances. One can say that the drill sergeant is at least training you in how to save your life in combat, and the imperious professor is at least preparing you for the vicissitudes of the real world. But to be given negative feedback by software — any software — is an insult. The drill sergeant and professor are at least human and have bona fide experience and merit. But to be told by software that you have failed is humiliating and degrading. Users, quite justifiably, hate to be humiliated and degraded. There is nothing that takes place inside a computer that is so important that it can justify humiliating or degrading a human user. We only resort to negative feedback out of habit.

Improving Error Messages: The Last Resort

Now we will discuss some methods of improving the quality of error message boxes, if indeed we are stuck using them. Use these recommendations only as a last resort, when you run out of other options.

A well-formed error message box should conform to these requirements:

Be polite

Be illuminating

Be helpful

Never forget that an error message box is the program reporting on its failure to do its job, and it is interrupting the user to do this. The error message box must be unfailingly polite. It must never even hint that the user caused this problem, because that is simply not true from the user's perspective. The customer is always right.

The user may indeed have entered some goofy data, but the program is in no position to argue and blame. It should do its best to deliver to the user what he asked for, no matter how silly. Above all, the program must not, when the user finally discovers his

silliness, say, in effect, "Well, you did something really stupid, and now you can't recover. Too bad." It is the program's responsibility to protect the user even when he takes inappropriate action. This may seem draconian, but it certainly isn't the user's responsibility to protect the computer from taking inappropriate action.

The error message box must illuminate the problem for the user. This means that it must give him the kind of information he needs to make an appropriate determination to solve the program's problem. It needs to make clear the scope of the problem, what the alternatives are, what the program will do as a default, and what information was lost, if any. The program should treat this as a confession, telling the user everything.

It is wrong, however, for the program to just dump the problem on the user's lap and wipe its hands of the matter. It should directly offer to implement at least one suggested solution right there on the error message box. It should offer buttons that will take care of the problem in various ways. If a printer is missing, the message box should offer options for deferring the printout or selecting another printer. If the database is hopelessly trashed and useless, it should offer to rebuild it to a working state, including telling the user how long that process will take and what side effects it will cause.

Figure shows an example of a reasonable error message. Notice that it is polite, illuminating, and helpful. It doesn't even hint that the user's behavior is anything but impeccable.

© Copyright Virtual University of Pakistan



42.3 Notifying and Confirming

Now, we discuss alert dialogs (also known as notifiers) and confirmation dialogs, as well as the structure of these interactions, the underlying assumptions about them, and how they, too, can be eliminated in most cases. ?

42.4 Alerts and Confirmations

Like error dialogs, alerts and confirmations stop the proceedings with idiocy, but they do not report malfunctions. An alert notifies the user of the program's action, whereas a confirmation also gives the user the authority to override that action. These dialogs pop up like weeds in most programs and should, much like error dialogs, be eliminated in favor of more useful idioms.

Alerts: Announcing the obvious

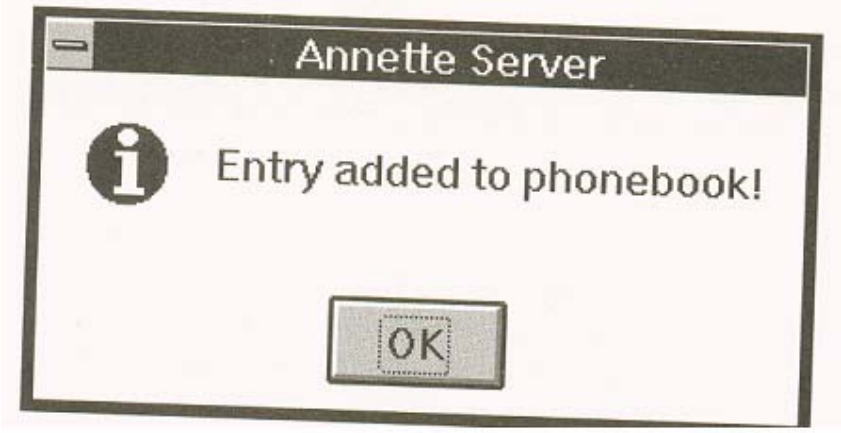
When a program exercises authority that it feels uncomfortable with, it takes steps to inform the user of its actions. This is called an alert. Alerts violate the axiom: A dialog box is another room; you should have a good reason to go. Even if an alert is justified (it seldom is), why go into another room to do it? If the program took some indefensible action, it should confess to it in the same place where the action occurred and not in a separate dialog box.

Conceptually, a program should either have the courage of its convictions or it should not take action without the user's direct guidance. If the program, for example, saves the user's file to disk automatically, it should have the confidence to know that it is doing the right thing. It should provide a means for the user to find out what the program did, but it doesn't have to stop the proceedings with idiocy to do so. If the program really isn't sure that it should save the file, it shouldn't save the file, but should leave that operation up to the user.

Conversely, if the user directs the program to do something — dragging a file to the trash can. for example — it doesn't need to stop the proceedings with idiocy to announce that the user just dragged a file to the trashcan. The program should ensure that there is

© Copyright Virtual University of Pakistan

adequate visual feedback regarding the action; and if the user has actually made the gesture in error, the program should silently offer him a robust Undo facility so he can backtrack.
The rationale for alerts is that they inform the user. This is a desirable objective, but not at the expense of smooth interaction flow.
Alerts are so numerous because they are so easy to create. Most languages offer some form of message box facility in a single line of code. Conversely, building an animated status display into the face of a program might require a thousand or more lines of code. Programmers cannot be expected to make the right choice in this situation. They have a conflict of interest, so designer: must be sure to specify precisely where information is reported on the surface of an application The designers must then follow up to be sure that the design wasn't compromised for the sake of rapid coding. Imagine if the contractor on
a building site decided unilaterally not to add a bathroom because it was just too much trouble to deal with the plumbing. There would be consequences.
Software needs to keep the user informed of its actions. It should have visual indicators built into its main screen to make such status information available to the user, should he desire it. Launching an alert to announce an unrequested action is bad enough. Putting up an alert to announce a requested action is pathological.
Software needs to be flexible and forgiving, but it doesn't need to be fawning and obsequious. The dialog box shown in Figure below is a classic example of an alert that should be put out of our misery. It announces that it added the entry to our phone book. This occurs immediately after we told it to add the entry to our phone book, which happened milliseconds after we physically added the entry to what appears to be our phone book. It stops the proceedings to announce the obvious.



It's as though the program wants approval for how hard it worked: "See, dear, I've cleaned your room for you. Don't you love me?" If a person interacted with us like this, we'd suggest that they seek counseling.

Confirmations S

When a program does not feel confident about its actions, it often asks the user for approval with a dialog box. This is called a confirmation, like the one shown in Figure below. Sometimes the confirmation is offered because the program second-guesses one of the user's actions. Sometimes the program feels that is not competent to make a decision it faces and uses a confirmation to give the user the choice instead. Confirmations always

© Copyright Virtual University of Pakistan

come from the program and never from the user. This means that they are a reflection of the implementation model and are not representative of the user's goals.



Remember, revealing the implementation model to users is a sure-fire way to create an inferior user interface. This means that confirmation messages are inappropriate. Confirmations get written into software when the programmer arrives at an impasse in his coding. Typically, he realizes that he is about to direct the program to take some bold action and feels unsure about taking responsibility for it. Sometimes the bold action is based on some conciliation the program detects, but more often it is based on a command the user issues. Typically, confirmation will be launched after the user issues a command that is either irrecoverable whose results might cause undue alarm.

Confirmations pass the buck to the user. The user trusts the program to do its job, and the program should both do it and ensure that it does it right. The proper solution is to make the action easily reversible and provide enough modeless feedback so that the user is not taken off-guard.

As a program's code grows during development, programmers detect numerous situations where they don't feel that they can resolve issues adequately. Programmers will unilaterally insert buck-passing code in these places, almost without noticing it. This tendency needs to be closely watched, because programmers have been known to insert dialog boxes into the code even after the user interface specification has been agreed upon. Programmers often don't consider confirmation dialogs to be part of the user interface, but they are.

THE DIALOG THAT CRIED, "WOLF!"

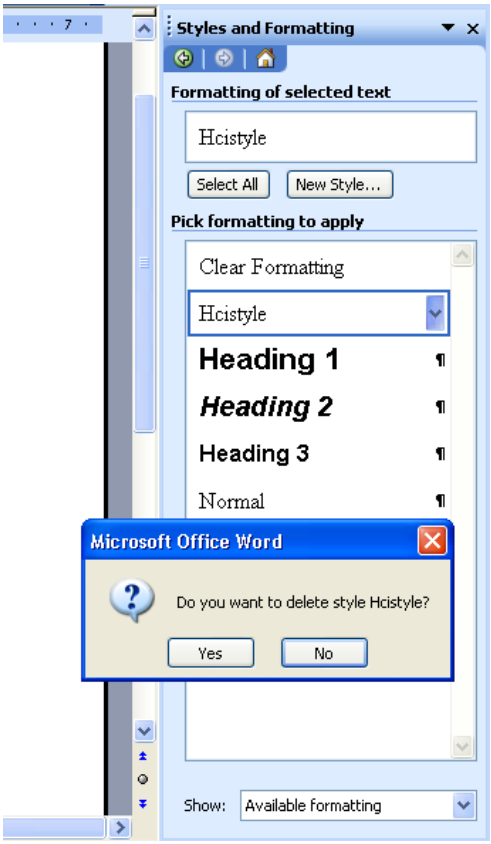
Confirmations illustrate an interesting quirk of human behavior: They only work when they are unexpected. That doesn't sound remarkable until you examine it in context. If confirmations are offered in routine places, the user quickly becomes inured to them and routinely dismisses them without a glance. The dismissing of confirmations thus becomes as routine as the issuing of them. If, at some point, a truly unexpected and dangerous situation arises — one that should be brought to the user's attention — he will, by rote, dismiss the confirmation, exactly because it has become routine. Like the fable of the boy who cried, "Wolf," when there is finally real danger, the confirmation box won't work because it cried too many times when there was no danger.

For confirmation dialog boxes to work, they must only appear when the user will almost definitely click the No or Cancel button, and they should never appear when the user is likely to click the Yes or OK button. Seen from this perspective, they look rather pointless, don't they?

The confirmation dialog box shown in Figure below is a classic. The irony of the confirmation dialog box in the figure is that it is hard to determine which styles to delete and which to keep. If the confirmation box appeared whenever we attempted to delete a style that was currently in use, it would at least then be helpful because the confirmation would be less routine. But why not instead put an icon next to the names of styles that are

© Copyright Virtual University of Pakistan

in use and dispense with the confirmation? The interface then provides more pertinent status information, so one can make a more informed decision about what to delete.



42.5 ELIMINATING CONFIRMATIONS

Three axioms tell us how to eliminate confirmation dialog boxes. The best way is to obey the simple dictum: Do, don't ask. When you design your software, go ahead and give it the force of its convictions (backed up by user research). Users will respect its brevity and its confidence.

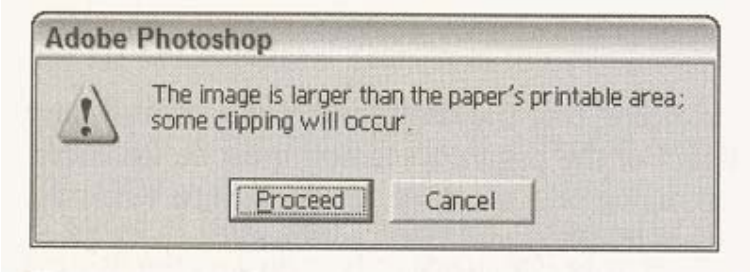
Of course, if the program confidently does something that the user doesn't like, it must have the capability to reverse the operation. Every aspect of the program's action must be undoable. Instead of asking in advance with a confirmation dialog box, on those rare occasions when the programs actions were out of turn, let the user issue the Stop-and-Undo command.

Most situations that we currently consider UN-protect able by Undo can actually be protected fairly well. Deleting or overwriting a file is a good example. The file can be moved to a suspense directory where it is kept for a month or so before it is physically deleted. The Recycle Bin in Windows uses this strategy, except for the part about automatically erasing files after a month: Users still have to manually take out the garbage.

Even better than acting in haste and forcing the user to rescue the program with Undo, you can make sure that the program offers the user adequate information so that the he never purposely issues a command that leads to an inappropriate action (or never omits a necessary command). The program should use sufficiently rich visual feedback so that the user is constantly kept informed, the same way the instruments on dashboards keep us informed of the state of our cars.

© Copyright Virtual University of Pakistan

Occasionally, a situation arises that really can't be protected by Undo. Is this a legitimate case for a confirmation dialog box? Not necessarily. A better approach is to provide users with protection the way we give them protection on the freeway: with consistent and clear markings. You can often build excellent, modeless warnings right into the interface. For instance, look at the dialog from Adobe Photoshop in Figure below, telling us that our document is larger than the available print area. Why has the program waited until now to inform us of this fact? What if guides were visible on the page at all times (unless the user hid them) showing the actual printable region? What if those parts of the picture outside the printable area were highlighted when the user moused over the Print button in the toolbar? Clear, modeless feedback is the best way to address these problems.



Much more common than honestly irreversible actions are those actions that are easily reversible but still uselessly protected by routine confirmation boxes. There is no reason whatsoever to ask for confirmation of a move to the Recycle Bin. The sole reason that the Recycle Bin exists is to implement an undo facility for deleted files.

42.6 Replacing Dialogs: Rich Modeless Feedback

Most computers now in use in the both the home and the office come with high-resolution displays and high-quality audio systems. Yet, very few programs (outside of games) even scratch the surface of using these facilities to provide useful information to the user about the status of the program, the users' tasks, and the system and its peripherals in general. It is as if an entire toolbox is available to express information to users, but programmers have stuck to using the same blunt instrument — the dialog — to communicate information. Needless to say, this means that subtle status information is simply never communicated to users at all, because even the most clueless designers know that you don't want dialogs to pop up constantly. But constant feedback is exactly what users need. It's simply the channel of communication that needs to be different.

In this section, well discuss rich modeless feedback, information that can be provided to the user in the main displays of your application, which don't stop the flow of the program or the user, and which can all but eliminate pesky dialogs.

42.7 Rich visual modeless feedback

Perhaps the most important type of modeless feedback is rich visual modeless feedback (RVMF). This type of feedback is rich in terms of giving in-depth information about the status or attributes of a process or object in the current application. It is visual in that it makes idiomatic use of pixels on the screen (often dynamically), and it is modeless in that this information is always readily displayed, requiring no special action or mode shift on the part of the user to view and make sense of the feedback.

For example, in Windows 2000 or XP, clicking on an object in a file manager window automatically causes details about that object to be displayed on the left-hand side of the file manager window. (In XP, Microsoft ruined this slightly by putting the information at

© Copyright Virtual University of Pakistan

the bottom of a variety of other commands and links. Also, by default, they made the Details area a drawer that you must open, although the program, at least, remembers its state.) Information includes title, type of document, its size, author, date of modification, and even a thumbnail or miniplayer if it is an image or media object. If the object is a disk, it shows a pie chart and legend depicting how much space is used on the disk. Very handy indeed! This interaction is perhaps slightly modal because it requires selection of the object, but the user needs to select objects anyway. This functionality handily eliminates the need for a properties dialog to display this information. Although most of this information is text, it still fits within the idiom.

© Copyright Virtual University of Pakistan

Lecture 43.

Information Retrieval

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Discuss how to communicate
- Learn how to retrieve the information

43.1 Audible feedback

In data-entry environments, clerks sit for hours in front of computer screens entering data. These users may well he examining source documents and typing by touch instead of looking at the screen. If a clerk enters something erroneous, he needs to be informed of it via both auditory and visual feedback. The clerk can then use his sense of hearing to monitor the success of his inputs while he keeps his eyes on the document.

The kind of auditory feedback we're proposing is *not* the same as the beep that accompanies an error message box. In fact, it isn't beep at all. The auditory indicator we propose as feedback for a problem is *silence.* The problem with much current audible feedback is the still-prevalent idea that, rather than positive audible feedback, negative feedback is desirable.

NEGATIVE AUDIBLE FEEDBACK: ANNOUNCING USER FAILURE

People frequently counter the idea of audible feedback with arguments that users don't like it. Users are offended by the sounds that computers make, and they don't like to have their computer beeping at them. This is likely true based on how computer sounds are widely used today — people have been conditioned by these unfortunate facts:

- Computers have always accompanied error messages with alarming noises.
- Computer noises have always been loud, monotonous and unpleasant.

Emitting noise when something bad happens is called negative audible feedback. On most systems, error message boxes are normally accompanied by loud, shrill, tinny "beeps," and audible feedback has thus become strongly associated them. That beep is a public announcement of the user's failure. It explains to all within earshot that you have done something execrably stupid. It is such a hateful idiom that most software developers now have an unquestioned belief that audible feedback is bad and should never again be considered as a part of interface design. Nothing could be further from the truth. It is the negative aspect of the feedback that presents problems, not the audible aspect.

Negative audible feedback has several things working against it. Because the negative feedback is issued at a time when a problem is discovered, it naturally takes on the characteristics of an alarm. Alarms are designed to be purposefully loud, discordant, and disturbing. They are supposed to wake sound sleepers from their slumbers when their house is on fire and their lives are at stake. They are like insurance because we all hope that they will never be heard. Unfortunately, users are constantly doing things that programs can't handle, so these actions have become part of the nor mal course of interaction. Alarms have

© Copyright Virtual University of Pakistan

no place in this normal relationship, the same way wt don't expect our car alarms to go off whenever we accidentally change lanes without using our turn indicators. Perhaps the most damning aspect of negative audible feedback is the implication that success must be greeted with silence. Humans like to know when they are

doing well. They *need* to know when they are doing poorly, but that doesn't mean that they like to hear about it. Negative feedback systems are simply appreciated less than positive feedback systems.

Given the choice of no noise versus noise for negative feedback, people will choose the former. Given the choice of no noise versus unpleasant noises for positive feedback, people will choose based on their personal situation and taste. Given the choice of no noise versus soft and pleasant noises for positive feedback, however, people will choose the feedback. We have never given our users a chance by putting high-quality, positive audible feedback in our programs, so it's no wonder that people associate sound with bad interfaces.

POSITIVE AUDIBLE FEEDBACK

Almost every object and system outside the world of software offers sound to indicate success rather than failure. When we close the door, we know that it is latched when we hear the click, but silence tells us that it is not yet secure. When we converse with someone and they say, "Yes" or "Uh-huh," we know that they have, at least minimally, registered what was said. When they are silent, however, we have reason to believe that something is amiss. When we turn the key in the ignition and get silence, we know we've got a problem. When we flip the switch on the copier and it stays coldly silent instead of humming, we know that we've got trouble. Even most equipment that we consider silent makes some noise: Turning on the stovetop returns a hiss of gas and a gratifying "whoomp" as the pilot ignites the burner. Electric ranges are inherently less friendly and harder to use because they lack that sound — they require indicator lights to tell us of their status.

When success with our tools yields a sound, it is called positive audible feedback. Our software tools are mostly silent; all we hear is the quiet click of the keyboard. Hey! That's positive audible feedback. Every time you press a key, you hear a faint but positive sound. Keyboard manufacturers could make perfectly silent keyboards, but they don't because we depend on audible feedback to tell us how we are doing. The feedback doesn't have to be sophisticated — those clicks don't tell us much — but they must be consistent. If we ever detect silence, we know that we have failed to press the key. The true value of positive audible feedback is that its absence is an extremely effective problem indicator.

The effectiveness of positive audible feedback originates in human sensitivity. Nobody likes to be told that they have failed. Error message boxes are negative feedback, telling the user that he has done something wrong. Silence can ensure that the user knows this without actually being told of the failure. It is remarkably effective, because the software doesn't have to insult the user to accomplish its ends.

Our software should give us constant, small, audible cues just like our keyboards. Our programs would be much friendlier and easier to use if they issued barely audible but easily identifiable sounds when user actions are correct. The program could issue an upbeat tone every time the user enters valid input to a field. If the program doesn't understand the input, it would remain silent and the user would be immediately informed of the problem and be able to correct his input without embarrassment or ego-bruising. Whenever the user starts to drag an icon, the computer could issue a low-volume sound reminiscent of sliding as the object is dragged. When it

© Copyright Virtual University of Pakistan

is dragged over pliant areas, an additional percussive tap could indicate this collision. When the user finally releases the mouse button, he is rewarded with a soft, cheerful "plonk" from the speakers for a success or with silence if the drop was not meaningful.

As with visual feedback, computer games tend to excel at positive audio feedback. Mac OS 9 also does a good job with subtle positive audio feedback for activities like documents saves and drag and drop. Of course, the audible feedback must be at the right volume for the situation. Windows and the Mac offer a standard volume control, so one obstacle to beneficial audible feedback has been overcome.

Rich modeless feedback is one of the greatest tools at the disposal of interaction designers. Replacing annoying, useless dialogs with subtle and powerful modeless communication can make the difference between a program users will despise and one they will love. Think of all the ways you might improve your own applications with RVMF and other mechanisms of modeless feedback!

43.2 Other Communication with Users

This Part of the lecture is about communicating with your users, and we would be remiss if we did not discuss ways of communicating to the user that are not only helpful to them, but which are also helpful to you, as creator or publisher of software, in asserting your brand and identity. In the best circumstances, these communications are not at odds, and this chapter presents recommendations that will enable you to make the most out of both aspects of user communication.

Your Identity on the Desktop

The modern desktop screen is getting quite crowded. A typical user has a half-dozen programs running concurrently, and each program must assert its identity. The user needs to recognize your application when he has relevant work to be done, and you should get the credit you deserve for the program you have created. There are several conventions for asserting identity in software.

Your program's name

By convention, your program's name is spelled out in the title bar of the program's main window. This text value is the program's title string, a single text value within the program that is usually owned by the program's main window. Microsoft Windows introduces some complications. Since Windows 95, the title string has played a greater role in the Windows interface. Particularly, the title string is displayed on the program's launch button on the taskbar.

The launch buttons on the taskbar automatically reduce their size as more buttons are added, which happens as the number of open programs increases. As the buttons get shorter, their title strings are truncated to fit.

Originally, the title string contained only the name of the application and the company brand name. Here's the rub: If you add your company's name to your program's name,

like, say "Microsoft Word," you will find that it only takes seven or eight running programs or open folders to truncate your program's launch-button string to "Microsoft." If you are also running "Microsoft Excel," you will find two adjacent buttons with identical, useless title strings. The differentiating portions of their names — "Word" and "Excel" are hidden.

The title string has, over the years, acquired another purpose. Many programs use it to display the name of the currently active document. Microsoft's Office Suite programs do this. In Windows 95, Microsoft appended the name of the active document to the right end of the title string, using a hyphen to separate it from the program name. In subsequent releases,

© Copyright Virtual University of Pakistan

Microsoft has reversed that order: The name of the document comes first, which is certainly a more goal-directed choice 41 as far as the user is concerned. The technique isn't a standard: but because Microsoft does it, it is often copied. It makes the title string extremely long, far too long to fit onto a launch button-but ToolTips come to the rescue!
What Microsoft could have done instead was add a new title string to the program's internal data structure. This string would be used only on the launch button (on the Taskbar), leaving the original title string for the window's title bar. This enables the designer and programmer to tailor the launch-button string for its restricted space, while letting the title string languish full-length on the always-roomier title bar.

Your program's icon

The second biggest component of your program's identity is its icon. There are two icons to worry about in Windows: the standard one at 32 pixels square and a miniature one that is 16 pixels square. Mac OS 9 and earlier had a similar arrangement; icons in OS X can theoretically be huge — up to 128x128 pixels. Windows XP also seems to make use of a 64x64 pixel, which makes sense given that screen resolutions today can exceed 1600x1200 pixels.
The 32x32 pixel size is used on the desktop, and the 16x16 pixel icon is used on the title bar, the taskbar, the Explorer, and at other locations in the Windows interface. Because of the increased importance of visual aesthetics in contemporary GUIs, you must pay greater attention to the quality of your program icon. In particular, you want your program's icon to be readily identifiable from a distance — especially the miniature version. The user doesn't necessarily have to be able to recognize it outright — although that would be nice — but he should be able to readily see that it is different from other icons.
Icon design is a craft unto itself, and is more difficult to do well than it may appear. Arguably, Susan Kare's design of the original Macintosh icons set the standard in the industry. Today, many visual interface designers specialize in the design of icons, and any applications will benefit from talent and experience applied to the effort of icon design.

Ancillary Application Windows

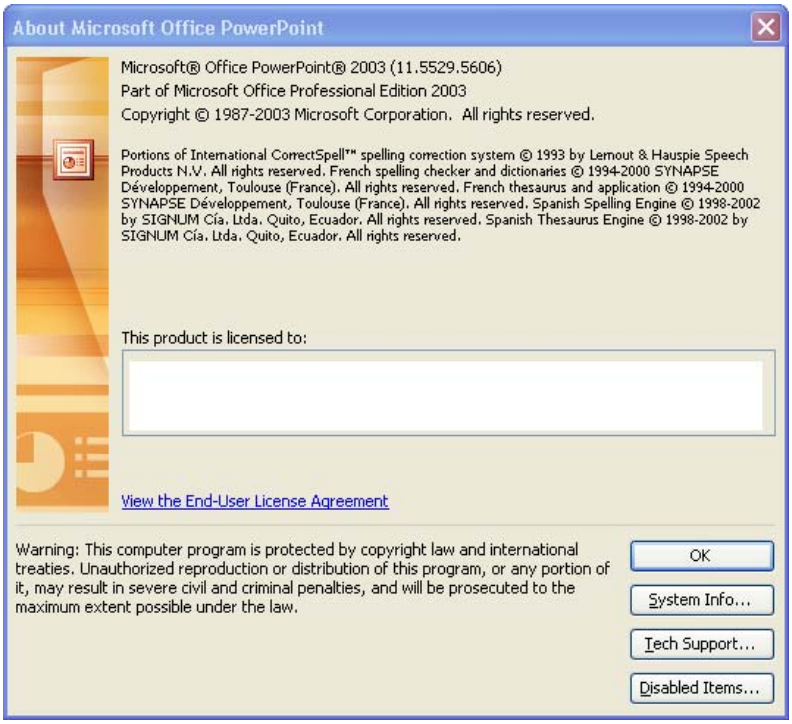
Ancillary application windows are windows that are not really part of the application's functionality, but are provided as a matter of convention. These windows are either available only on request or are offered up by the program only once, such as the
programs credit screen. Those that are offered unilaterally by the program are erected when the program is used for the *very* first time or each time the program is initiated. All these windows, however, form channels of communication that can both help the user and better communicate your brand.

About boxes

The About box is a single dialog box that — by convention — identifies the program to the user. The About box is also used as the program's credit screen, identifying the people who created it. Ironically, the About box rarely tells the user much about the program. On the Macintosh, the About box can be summoned from the top of the Apple pop-up menu. In Windows, it is almost always found at the bottom of the Help menu.
Microsoft has been consistent with About boxes in its programs, and it has taken a simple approach to its design, as you can see in Figure below. Microsoft uses the About box almost exclusively as a place for identification, a sort of driver's license for software. This is unfortunate, as it is a good place to give the curious user an overview of the program in a way that doesn't intrude on those users who don't need it. It is often, but not always, a

© Copyright Virtual University of Pakistan

good thing to follow in Microsoft's design footsteps. This is one place where diverging from Microsoft can offer a big advantage.



The main problem with Microsoft's approach is that the About box doesn't tell the user about' the program. In reality, it is an identification box. It identifies the program by name and version number. It identifies various copyrights in the program. It identifies the user and the user's company. These are certainly useful functions, but are more useful for Microsoft customer support than for the user.

The desire to make About boxes more useful is clearly strong — otherwise, we wouldn't see, memory usage and system-information buttons on them. This is admirable, but, by taking a more ... goal-directed approach, we can add information to the About box that really can help the user. The single most important thing that the About box can convey to the user is the scope of the program. It should tell, in the broadest terms, what the program can and can't do. It should also state succinctly what the program does. Most program authors forget that many users don't have any idea what the InfoMeister 3000 Version 4.0 program actually does. This is the place to gently clue ,; them in.

The About box is also a great place to give the one lesson that might start a new user success fully. For example, if there is one new idiom — like a direct-manipulation method — that is critical to the user interaction, this is a good place to briefly tell him about it. Additionally, the About box can direct the new user to other sources of information that will help him get his bearings in the program.

Because the current design of this facility just presents the program's fine print instead of telling the *user about* the program, it should be called an Identity box instead of an About box, and that's how well refer to it from this point on. The Identity box identifies the program to the user, " and the dialog in Figure above fulfills this definition admirably. It tells us all the stuff

© Copyright Virtual University of Pakistan

the lawyers require and the tech support people need to know. Clearly, Microsoft has made the decision that an Identity box is important, whereas a true About box is expendable.

As we've seen, the Identity box must offer the basics of identification, including the publisher's name, the program's icons, the program's version number, and the names of its authors. Another item that could profitably be placed here is the publisher's technical support telephone number.

Many software publishers don't identify their programs with sufficient discrimination to tie , them to a specific software build. Some vendors even go so far as to issue the same version number to significantly different programs for marketing reasons. But the version number in the Identity — or About — box is mainly used by customer support. A misleading version number will cost the publisher a significant amount of phone-support time just figuring out precisely which version of the program the user has. It doesn't matter what scheme you use, as long as this number is very specific.

The About box (not the Identity box) is absolutely the right place to state the product team's names. The authors firmly believe that credit should be given where credit is due in the design, development, and testing of software. Programmers, designers, managers, and testers all deserve to see their names in lights. Documentation writers sometimes get to put their names in the manual, but the others only have the program itself. The About box is one of the few dialogs that has no functional overlap with the main program, so there is no reason why it can't be oversized. Take the space to mention everyone who contributed. Although some programmers are indifferent to seeing their names on the screen, many programmers are powerfully motivated by it and really appreciate managers who make it happen. What possible reason could there be for *not* naming the smart, hard-working people who built the program?

This last question is directed at Bill Gates (as it was in the first edition in 1995), who has a corporate-wide policy that individual programmers *never* get to put their names in the About boxes of programs. He feels that it would be difficult to know where to draw the line with individuals. But the credits for modern movies are indicative that the entertainment industry, for one, has no such worries. In fact, it is in game software that development credits are most often featured. Perhaps now that Microsoft is heavy into the game business things will change — but don't count on it.

Microsoft's policy is disturbing because its conventions are so widely copied. As a result, *Its* no-programmer-names policy is also widely copied by companies who have no real reason for it other than blindly following Microsoft.

Splash screens

A splash screen is a dialog box displayed when a program first loads into memory. Sometimes it may just be the About box or Identity box, displayed automatically, but more often publishers create a separate splash screen that is more engaging and visually exciting.

The splash screen should be placed on the screen as soon as user launches the program, so that he can view it while the bulk of the program loads and prepares itself for running. After a few seconds have passed, it should disappear and the program should go about its business. If, during the splash screen's tenure, the user presses any key or clicks any mouse button, the splash screen should disappear immediately. The program must show the utmost respect for the user's time, even if it is measured in milliseconds.

The splash screen is an excellent opportunity to create a good impression. It can be used to reinforce the idea that your users made a good choice by purchasing your product. It also

© Copyright Virtual University of Pakistan

helps to establish a visual brand by displaying the company logo, the product logo, the product icon, and other appropriate visual symbols.
Splash screens are also excellent tools for directing first-time users to training resources that are not regularly used. If the program has built-in tutorials or configuration options, the splash screen can provide buttons that take the user directly to these facilities (in this case, the splash screen should remain open until manually dismissed).
Because splash screens are going to be seen by first-timers, if you have something to say to them, this is a good place to do it. On the other hand, the message you offer to those first-timers will be annoying to experienced users, so subsequent instances of the splash screen should be more generic. Whatever you say, be clear and terse, not long-winded or cute. An irritating message on the splash screen is like a pebble in your shoe, rapidly creating a sore spot if it isn't removed promptly.

Shareware splash screens

If your program is shareware, the splash screen can be your most important dialog (though not your users'). It is the mechanism whereby you inform users of the terms of use and the appropriate way to pay for your product. Some people refer to shareware splash screens as the guilt screen. Of course, this information should also be embedded in the program where the user can request it, but by presenting to users each time the program loads, you can reinforce the concept that the program *should* be paid for. On the other hand, there's a fine line you need to tread lest your sales pitch alienate users. The best approach is to create an excellent product, not to guilt-trip potential customers.

Online help

Online help is just like printed documentation, a reference tool for perpetual intermediates. Ultimately, online help is not important, the way that the user manual of your car is not important. If you find yourself needing the manual, it means that your car is badly designed. The design is what is important.
A complex program with many features and functions should come with a reference document: a place where users who wish to expand their horizons with a product can find definitive answers. This document can be a printed manual or it can be online help. The printed manual is comfortable, browsable, friendly, and can be carried around. The online help is searchable, semi-comfortable, very lightweight, and cheap.

The index

Because you don't read a manual like a novel, the key to a successful and effective reference document is the quality of the tools for finding what you want in it. Essentially, this means the index. A printed manual has an index in the back that you use manually. Online help has an automatic index search facility.
The experts suspect that few online help facilities they've seen were indexed by a professional indexer. However many entries are in your program's index, you could probably double the number. What's more, the index needs to be generated by examining the program and all its features, not by examining the help text This is not easy, because it demands that a highly skilled indexer be intimately familiar with all the features of the program. It may be easier to rework the interface to improve it than to create a really good index.
The list of index entries is arguably more important than the text of the entries themselves. The user will forgive a poorly written entry with more alacrity than he will forgive a missing entry. The index must have as many synonyms as possible for topics. Prepare for it to be huge. The user who needs to solve a problem will be thinking "How do I turn this

© Copyright Virtual University of Pakistan

cell black?" not "How can I set the *shading* of this cell to 100%?" If the entry is listed under shading, the index fails the user. The more goal-directed your thinking is, the better the index will map to what might possibly pop into the user's head when he is looking for something. One index model that works is the one in *The Joy of Cooking,* Irma S. Rombaur & Marion Rombaur Becker (Bobbs-Merrill, 1962). That index is one of the most complete and robust of any the authors have used,

Shortcuts and overview

One of the features missing from almost every help system is a shortcuts option. It is an item in the Help menu which when selected, shows in digest form all the tools and keyboard commands for the program's various features. It is a very necessary component on any online help system because it provides what perpetual intermediates need the most: access to features. They need the tools and commands more than they need detailed instructions.

The other missing ingredient from online help systems is overview. Users want to know how the Enter Macro command works, and the help system explains uselessly that it is the facility that lets you enter macros into the system. What we need to know is scope, effect, power, upside, downside, and why we might want to use this facility both in absolute terms and in comparison to similar products from other vendors. @Last Software provides online streaming video tutorials for its architectural sketching application, SketchUp. This is a fantastic approach to overviews, particularly if they are also available on CD-ROM.

Not for beginners

Many help systems assume that their role is to provide assistance to beginners. This is not true. Beginners stay away from the help system because it is generally just as complex as the program. Besides, any program whose basic functioning is too hard to figure out just by experimentation is unacceptable, and no amount of help text will resurrect it. Online help should ignore first-time users and concentrate on those people who are already successfully using the product, but who want to expand their horizons: the perpetual intermediates.

Modeless and interactive help

ToolTips are modeless online help, and they are incredibly effective. Standard help systems, on the other hand, are implemented in a separate program that covers up most of the program for which it is offering help. If you were to ask a human how to perform a task, he would use his finger to point to objects on the screen to augment his explanation. A separate help program that obscures the main program cannot do this. Apple has used an innovative help system that directs the user through a task step by step by highlighting menus and buttons that the user needs to activate in sequence. Though this is not totally modeless, it is interactive and closely integrated with the task the user wants to perform, and not a separate room, like reference help systems,

Wizards

Wizards are an idiom unleashed on the world by Microsoft, and they have rapidly gained popularity among programmers and user interface designers. A wizard attempts to guarantee success in using a feature by stepping the user through a series of dialog boxes. These dialogs parallel a complex procedure that is "normally" used to manage a feature of the program. For example, a wizard helps the user create a presentation in PowerPoint.

Programmers like wizards because they get to treat the user like a peripheral device. Each of the wizard's dialogs asks the user a question or two, and in the end the program performs

© Copyright Virtual University of Pakistan

whatever task was requested. They are a fine example of interrogation tactics on the program's part, and violate the axiom: A*sking questions isn't the same as providing choices.*
Wizards are written as step-by-step procedures, rather than as informed conversations between user and program. The user is like the conductor of a robot orchestra, swinging the baton to set the tempo, but otherwise having no influence on the proceedings. In this way, wizards rapidly devolve into exercises in confirmation messaging. The user learns that he merely clicks the Next button on each screen without critically analyzing why.

There is a place for wizards in actions that are very rarely used, such as installation and initial configuration. In the weakly interactive world of HTML, they have also become the standard idiom for almost all transactions on the Web — something that better browser technology will eventually change.
A better way to create a wizard is to make a simple, automatic function that asks no questions of the user but that just goes off and does the job. If it creates a presentation, for example, it should create it, and then let the user have the option, later, using standard tools, to change the presentation. The interrogation tactics of the typical wizard are not friendly, reassuring, or particularly helpful. The wizard often doesn't explain to the user what is going on.
Wizards were purportedly designed to improve user interfaces, but they are, in many cases, having the opposite effect. They are giving programmers license to put raw implementation model interfaces on complex features with the bland assurance that: "We'll make it easy with a wizard." This is all too reminiscent of the standard abdication of responsibility to users: "We'll be sure to document it in the manual."

"Intelligent" agents

Perhaps not much needs to be said about Clippy and his cousins, since even Microsoft has turned against their creation in its marketing of Windows XP (not that it has actually *removed* Clippy from XP, mind you). Clippy is a remnant of research Microsoft did in the creation of BOB, an "intuitive" real-world, metaphor-laden interface remarkably similar to General Magic's Magic Cap interface. BOB was populated with anthropomorphic, animated characters that conversed with users to help them accomplish things. It was one of Microsoft's most spectacular interface failures. Clippy is a descendant of these help agents and is every bit as annoying as they were.
A significant issue with "intelligent" animated agents is that by employing animated anthropomorphism, the software is upping the ante on user expectations of the agent's intelligence. If it can't deliver on these expectations, users will quickly become furious, just as they would with a sales clerk in a department store who claims to be an expert on his products, but who, after a few simple questions, proves himself to be clueless.
These constructs soon become cloying and distracting. Users of Microsoft Office are trying to accomplish something, not be entertained by the antics and pratfalls of the help system. Most applications demand more direct, less distracting, and trust worthier means of getting assistance.

43.3 Improving Data Retrieval

In the physical world, storing and retrieving are inextricably linked; putting an item on a shelf (storing it) also gives us the means to find it later (retrieving it}. In the digital world, the only thing linking these two concepts is our faulty thinking. Computers will enable remarkably sophisticated retrieval techniques if only we are able to break our thinking out of its traditional box. This part of lecture discusses methods of data retrieval from an interaction standpoint and presents some more human-centered approaches to the problem of finding useful information.

© Copyright Virtual University of Pakistan

Storage and Retrieval Systems

A storage system is a method for safekeeping goods in a repository. It is a physical system composed of a container and the tools necessary to put objects in and take them back out again. A retrieval system is a method for finding goods in a repository. It is a logical system that allows the; goods to be located according to some abstract value, like name, position or some aspect of the; contents.

Disks and files are usually rendered in implementation terms rather than in accord with the user's mental model of how information is stored. This is also true in the methods we use for *finding* information after it has been stored. This is extremely unfortunate because the computer is the one tool capable of providing us with significantly better methods of finding information than those physically possible using mechanical systems. But before we talk about how to improve retrieval, let's briefly discuss how it works.

Storage and Retrieval in the Physical World

We can own a book or a hammer without giving it a name or a permanent place of residence in our houses. A book can be identified by characteristics other than a name — a color or a shape, for example. However, after we accumulate a large number of items that we need to find and use, it helps to be a bit more organized.

Everything in its place: Storage and retrieval by location

It is important that there be a proper place for our books and hammers, because that is how we find them when we need them. We can't just whistle and expect them to find us; we must know where they are and then go there and fetch them. In the physical world, the actual location of a thing is the means to finding it. Remembering where we put something- -its address — is vital both to finding it, and putting it away so it can be found again. When we want to find a spoon, for example, we go to the place where we keep our spoons. We don't find the spoon by referring to any inherent characteristic of the spoon itself. Similarly, when we look for a book, we either go to where we left the book, or we guess that it is stored with other books. We don't find the book by association. That is, we don't find the book by referring to its contents.

In this model, which works just fine in your home, the storage system is the same as the retrieval system: Both are based on remembering locations. They are coupled storage and retrieval systems.

Indexed retrieval

This system of everything in its proper place sounds pretty good, but it has a flaw: It is limited in scale by human memory. Although it works for the books, hammers, and spoons in your house, it doesn't work at all for the volumes stored, for example, in the Library of Congress.

In the world of hooks and paper on library shelves, we make use of another tool to help us find things: the Dewey Decimal system (named after its inventor, American philosopher and educator John Dewey). The idea was brilliant: Give even' book title a unique number based on its subject matter and title and shelve the books in this numerical order. If you know the number, you can easily find the book, and other books related to it by subject would be near by — perfect for research. The only remaining issue was how to discover the number for a given book. Certainly nobody could be expected to remember every number.

© Copyright Virtual University of Pakistan

The solution was an index, a collection of records that allows you to find the location of an item by looking up an attribute of the item, such as its name. Traditional library card catalogs provided lookup by three attributes: author, subject, and title. When the book is entered into the library system and assigned a number, three index cards are created for the book, including all particulars and the Dewey Decimal number. Each card is headed by the author's name, the subject, or the title. These cards are then placed in their respective indices in alphabetical order. When you want to find a book, you look it up in one of the indices and find its number. You then find the row of shelves that contains books with numbers in the same range as your target by examining signs. You search those particular shelves, narrowing your view by the lexical order of the numbers until you find the one you want.

You *physically* retrieve the book by participating in the system of storage, but you *logically* find the book you want by participating in a system of retrieval. The shelves and numbers are the storage system. The card indices are the retrieval system. You identify the desired book with one and fetch it with the other. In a typical university or professional library, customers are not allowed into the stacks. As a customer, you identify the book you want by using only the retrieval system. The librarian then fetches the book for you by participating only in the storage system. The unique serial number is the bridge between these two interdependent systems. In the physical world, both the retrieval system and the storage system may be very labor intensive. Particularly in older, non-computerized libraries, they are both inflexible. Adding a fourth index based on acquisition date, for example, would be prohibitively difficult for the library.

Storage and Retrieval in the Digital World

Unlike in the physical world of books, stacks, and cards, it's not very hard to add an index in the computer. Ironically, in a system where easily implementing dynamic, associative retrieval mechanisms is at last possible, we often don't implement any retrieval system. Astonishingly, we don't use indices at all on the desktop.

In most of today's computer systems, there is no retrieval system other than the storage system. If you want to find a file on disk you need to know its name and its place. It's as if we went into the library, burned the card catalog, and told the patrons that they could easily find what they want by just remembering the little numbers painted on the spines of the books. We have put 100 percent of the burden of file retrieval on the user's memory while the CPU just sits there idling, executing billions of nop instructions.

Although our desktop computers can handle hundreds of different indices, we ignore this capability and have no indices at all pointing into the files stored on our disks. Instead, we have to remember where we put our files and what we called them in order to find them again. This omission is one of the most destructive, backward steps in modern software design. This failure can be attributed to the interdependence of files and the organizational systems in which they exist, an interdependence that doesn't exist in the mechanical world.

Retrieval methods

There are three fundamental ways to find a document on a computer. You can find it by remembering where you left it in the file structure, by positional retrieval. You can find it by remembering its identifying name, by identity retrieval. The third method, associative or attributed-based retrieval, is based on the ability to search for a document based on some inherent quality of the document itself. For example, if you want to find a book with

© Copyright Virtual University of Pakistan

a red cover, or one that discusses light rail transit systems, or one that contains photographs of steam locomotives, or one that mentions Theodore Judah, the method you must use is associative.

Both positional and identity retrieval are methods that also function as storage systems, and on computers, which can sort reasonably well by name, they are practically one and the same. Associative retrieval is the one method that is not also a storage system. If our retrieval system is based solely on storage methods, we deny ourselves any attribute-based searching and we must depend on memory. Our user must know what information he wants and where it is stored in order to find it. To find the spreadsheet in which he calculated the amortization of his home loan he has to know that he stored it in the directory called Home and that it was called amoral. If he doesn't remember either of these facts, finding the document can become quite difficult

An attribute-based retrieval system

For early GUI systems like the original Macintosh, a positional retrieval system almost made sense: The desktop metaphor dictated it (you don't use an index to look up papers on your desk), and there were precious few documents that could be stored on a 144K floppy disk. However, our current desktop systems can now easily hold 250,000 times as many documents! Yet we still use the same metaphors and retrieval model to manage our data. We continue to render our software's retrieval systems in strict adherence to the implementation model of the storage system, ignoring the power and ease-of-use of a system for finding files that is distinct from the system for keeping files.

An attribute-based retrieval system would enable us to find our documents by their contents. For example, we could find all documents that contain the text string "superelevation". For such a search system to really be effective, it should know where all documents can be found, so the user doesn't have to say "Go look in such-and-such a directory and find all documents that mention "superelevation." This system would, of course, know a little bit about the domain of its search so it wouldn't try to search the entire Internet, for example, for "superelevation" unless we insist.

A well-crafted, attribute-based retrieval system would also enable the user to browse by synonym or related topics or by assigning attributes to individual documents. The user can then & dynamically define sets of documents having these overlapping attributes. For example, imagine a consulting business where each potential client is sent a proposal letter. Each of these letters is different and is naturally grouped with the files pertinent to that client. However, there is a definite relationship between each of these letters because they all serve the same function: proposing a business relationship. It would be very convenient if a user could find and gather up all such proposal letters while allowing each one to retain its uniqueness and association with its particular client. A file system based on place —on its single storage location — must of necessity store each document by a single attribute rather than multiple characteristics.

The system can learn a lot about each document just by keeping its eyes and ears open. If the attribute based retrieval system remembers some of this information, much of the setup burden on the user is made unnecessary. The program could, for example, easily remember such things as:

- The program that created the document
- The type of document: words, numbers, tables, graphics
- The program that last opened the document.
- If the document is exceptionally large or small
- If the document has been untouched for a long time

© Copyright Virtual University of Pakistan

- The length of time the document was last open
- The amount of information that was added or deleted during the last edit
- Whether or not the document has been edited by more than one type of program
- Whether the document contains embedded objects from other programs
- If the document was created from scratch or cloned from another
- If the document is frequently edited
- If the document is frequently viewed but rarely edited
- Whether the document has been printed and where
- How often the document has been printed, and whether changes were made to it each time immediately before printing
- Whether the document has been faxed and to whom
- Whether the document has been e-mailed and to whom

The retrieval system could find documents for the user based on these facts without the user ever having to explicitly record anything in advance. Can you think of other useful attributes the system might remember?

One product on the market provides much of this functionality for Windows. Enfish Corporation sells a suite of personal and enterprise products that dynamically and invisibly create an index of information on your computer system, across a LAN if you desire it (the Professional version), and even across the Web. It tracks documents, bookmarks, contacts, and e-mails — extracting all the reasonable attributes. It also provides powerful sorting and filtering capability. It is truly a remarkable set of products. We should all learn from the Enfish example.

There is nothing wrong with the disk file storage systems that we have created for ourselves. The only problem is that we have failed to create adequate disk file retrieval systems. Instead, we hand the user the storage system and call it a retrieval system. This is like handing him a bag of groceries and calling it a gourmet dinner. There is no reason to change our file storage systems. The Unix model is fine. Our programs can easily remember the names and locations of the files they have worked on, so they aren't the ones who need a retrieval system: It's for us human users.

© Copyright Virtual University of Pakistan

Lecture 44.

Emerging Paradigms

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the role of information architecture
- Understand the importance of accessibility

Metadata

A web site is a collection of interconnected systems with complex dependencies. A single link on a page can simultaneously be part of the site's structure, organization, labeling, navigation, and searching systems. It's useful to study these systems independently, but it's also crucial to consider how they interact. Reductionism will not tell us the whole truth.

Metadata and controlled vocabularies present a fascinating lens through which to view the network of relationships between systems. In many large metadata-driven web sites, controlled vocabularies have become the glue that holds the systems together. A thesaurus on the back end can enable a more seamless and satisfying user experience on the front end.

In addition, the practice of thesaurus design can help bridge the gap between past and present. The first thesauri were developed for libraries, museums, and government agencies long before the invention of the World Wide Web. As information architects we can draw upon these decades of experience, but we can't copy indiscriminately. The web sites and intranets we design present new challenges and demand creative solutions.
When it comes to definitions, metadata is a slippery fish. Describing it as "data about data" isn't very helpful. The following excerpt from Dictionary.com takes us 2 little further:
In data processing, meta-data is definitional data that provides information about or documentation of other data managed within an application or environment. For example, meta-data would document data about data elements or attributes (name, size, data type, etc) and data about records or data structures (length, fields, columns, etc) and data about data (where it is located, how it is associated, ownership, etc.).
Meta-data may include descriptive information about the context, quality and condi¬tion, or characteristics of the data.
While these tautological explanations could lead us into the realms of epistemology and metaphysics, we won't go there. Instead, let's focus on the role that metadata plays in the practical realm of information architecture.
Metadata tags are used to describe documents, pages, images, software, video and audio files, and other content objects for the purposes of improved navigation and retrieval. The HTML keyword meta tag used by many web sites provides a simple example. Authors can freely enter words and phrases that describe the content. These keywords are not displayed in the interface, but are available for use by search engines.

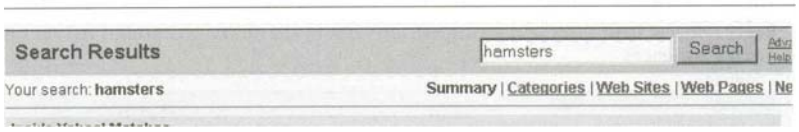
© Copyright Virtual University of Pakistan

<meta name="keywords" content="information architecture, content management, knowledge management, user experience">

Many companies today are using metadata in more sophisticated ways. Leveraging content management software and controlled vocabularies, they create dynamic meta data-driven web sites that support distributed authoring and powerful navigation. This metadata-driven model represents a profound change in how web sites are created and managed. Instead of asking, "Where do I place this document in the taxaonomy?" we can now ask, "How do I describe this document?" The software and vocabulary systems take care of the rest.

Controlled Vocabularies

Vocabulary control comes in many shapes and sizes. At its most vague, a controlled vocabulary is any defined subset of natural language. At its simplest, a controlled vocabulary is a list of equivalent terms in the form of a synonym ring, or a list of preferred terms in the form of an authority file. Define hierarchical relationships between terms (e.g., broader, narrower) and you've got a classification scheme. Model associative relationships between concepts (e.g., see also, see related) and
Since a full-blown thesaurus integrates all the relationships and capabilities of the simpler forms, let's explore each of these building blocks before taking a close look at the "Swiss Army Knife" of controlled vocabularies.
Classification schemes can also be used in the context of searching. Yahoo! does this very effectively. Yahoo!'s search results present "Category Matches," which reinforces users' familiarity with Yahoo!'s classification scheme.



Pel$:find Hamsters in Yahoo! Pets
Bizarre Humor> Hamster Dance
Humor> Hamsters
Rodents> Hamsters
List "hamsters"_by location

The above are Category Matches at Yahoo!
The important point here is that classification schemes are not tied to a single view or instance. They can be used on both the back end and the front end in all sorts of ways. We'll explore types of classification schemes in more detail later in this chapter, but first let's take a look at the "Swiss Army Knife" of vocabulary control, the thesaurus.

Thesauri

Dictionary.com defines thesaurus as a "book of synonyms, often including related and contrasting words and antonyms." This usage harkens back to our high school English classes, when we chose big words from the thesaurus to impress our teachers.
Our species of thesaurus, the one integrated within a web site or intranet to improve navigation and retrieval, shares a common heritage with the familiar reference text but

© Copyright Virtual University of Pakistan

has a different form and function. Like the reference book, our thesaurus is a semantic network of concepts, connecting words to their synonyms, homonyms, antonyms, broader and narrower terms, and related terms.
However, our thesaurus takes the form of an online database, tightly integrated with the user interface of a web site or intranet. And while the traditional thesaurus helps people go from one word to many words, our thesaurus does the opposite. Its most important goal is synonym management, the mapping of many synonyms or word variants onto one preferred term or concept, so the ambiguities of language don't prevent people from finding what they need.
So, for the purposes of this book, a thesaurus is:
A controlled vocabulary in which equivalence, hierarchical, and associative relationships are identified for purposes of improved retrieval..
--ANSI/NISO Z39.19 -1993 (R1998). Guidelines for the Construction, Format, and Management of Monolingual Thesauri.
A thesaurus builds upon the constructs of the simpler controlled vocabularies, modeling these three fundamental types of semantic relationships.
Each preferred term becomes the center of its own semantic network. The equivalence relationship is focused on synonym management. The hierarchical relationship enables the classification of preferred terms into categories and subcategories. The associative relationship provides for meaningful connections that are not handled by the hierarchical or equivalence relationships. All three relationships can be useful in different ways for the purposes of information retrieval and navigation.

44.1 Accessibility

Accessibility is a general term used to describe the degree to which a system is usable by as many people as possible without modification. It is not to be confused with usability which is used to describe how easily a thing can be used by any type of user. One meaning of accessibility specifically focuses on people with disabilities and their use of assistive devices such as screen-reading web browsers or wheelchairs. Other meanings are discussed below.
Accessibility is strongly related to universal design in that it is about making things as accessible as possible to as wide a group of people as possible. However, products marketed as having benefited from a Universal Design process are often actually the same devices customized specifically for use by people with disabilities. It is rare to find a Universally Designed product at the mass-market level that is used mostly by non-disabled people.
The disability rights movement advocates equal access to social, political and economic life which includes not only physical access but access to the same tools, organisations and facilities which we all pay for.
A typical sign for wheelchair accessibilityAccessibility is about giving equal access to everyone.
While it is often used to describe facilities or amenities to assist people with disabilities, as in "wheelchair accessible", the term can extend to Braille signage, wheelchair ramps, audio signals at pedestrian crossings, walkway contours, website design, and so on.
Various countries have legislation requiring physical accessibility:
In the UK, the Disability Discrimination Act 1995 has numerous provisions for accessibility.

© Copyright Virtual University of Pakistan

In the US, under the Americans with Disabilities Act of 1990, new public and private business construction generally must be accessible. Existing private businesses are required to increase the accessibility of their facilities when making any other renovations in proportion to the cost of the other renovations. The U.S. Access Board is "A Federal Agency Committed to Accessible Design for People with Disabilities." Many states in the US have their own disability laws.
In Ontario, Canada, the Ontarians with Disabilities Act of 2001 is meant to "improve the identification, removal and prevention of barriers faced by persons with disabilities..."

Introduction to Web Accessibility

Introduction
Most people today can hardly conceive of life without the Internet. It provides access to information, news, email, shopping, and entertainment. The Internet, with its ability to serve out information at any hour of the day or night about practically any topic conceivable, has become a way of life for an impatient, information-hungry generation. Some have argued that no other single invention has been more revolutionary since that of Gutenberg's original printing press in the mid 1400s. Now, at the click of a mouse, the world can be "at your fingertips"—that is, if you can use a mouse... and if you can see the screen... and if you can hear the audio—in other words, if you don't have a disability of any kind.
Before focusing on the challenges that people with disabilities face when trying to access web content, it makes more sense to discuss the ways in which the Internet offers incredible opportunities to people with disabilities that were never before possible. The web's potential for people with disabilities is truly remarkable.

The Web Offers Unprecedented Opportunities
The Internet is one of the best things that ever happened to people with disabilities. You may not have thought about it that way, but all you have to do is think back to the days before the Internet was as ubiquitous as it is today to see why this is so.

For example, without the Internet, how did blind people read newspapers? The answer is that they mostly didn't. At best, they could ask a family member or friend to read the newspaper to them. This method works, but it makes blind people dependent upon others. They could never read the newspaper themselves. You might think that audiotapes or Braille printouts of newspapers could offer a reasonable solution, but both options are expensive and slow compared to the rate at which publishers create and distribute newspapers. Blind people wouldn't receive the news until after it was no longer new. Not only that, but a Braille version of the Sunday New York Times would be so big and bulky with the extra large and thick Braille embossed paper that you'd practically have to use a forklift to move it around. None of these methods of reading newspapers are ideal. They're too slow, expensive, and too dependent upon other people.
With the advent of the World Wide Web, many newspapers now publish their content electronically in a format that can be read by text-to-speech synthesizer software programs (often called "screen readers") used by the blind. These software programs read text out loud so that blind people can use computers and access any text content through the computer. Suddenly, blind people don't have to rely on the kindness of other people to read the newspaper to them. They don't have to wait for expensive

© Copyright Virtual University of Pakistan

audio tapes or expensive, bulky Braille printouts. They simply open a web browser and listen to their screen reader as it reads the newspaper to them, and they do it when they want to do it. The Internet affords a whole new level of independence and opportunity to blind people. When you understand the impact that the Internet can have in the lives of blind people, the concept of web accessibility takes on a whole new level of significance.

Similarly, people with motor disabilities who cannot pick up a newspaper or turn its pages can access online newspapers through their computer, using certain assistive technologies that adapt the computer interface to their own disabilities. Sometimes the adaptations are crude, such as having the person place a stick in the mouth, and to use that stick to type keyboard commands. In other cases, the adaptations are more sophisticated, as in the use of eye-tracking software that allows people to use a computer with nothing more than eye movements. People with tremors may use a special keyboard with raised ridges in-between the keys so that they can place their hand down on the keyboard and then type the letters, rather than risk typing the wrong keys. Most of these people would not be able to use a mouse with much accuracy. Regardless of the level of sophistication, many of these adaptations have one thing in common: they make use of the keyboard, or emulate the use of a keyboard, rather than the use of a mouse. As with people who are blind, the Internet allows people with motor disabilities to access information in ways that they never could before.

People who are deaf always had the possibility of reading newspapers on their own, so it may seem that the Internet does not offer the same type of emancipation that it does to those who are blind or to those with motor disabilities, but there are a few cases in which the Internet can still have a large impact. For example, they can read online transcripts of important speeches, or view multimedia content that has been fully captioned.

Falling Short of the Web's Potential

Despite the Web's great potential for people with disabilities, this potential is still largely unrealized. Where can you find web-based video or multimedia content that has been fully captioned for the deaf? What if the Internet content is only accessible by using a mouse? What do people do if they can't use a mouse? And what if web developers use all graphics instead of text? If screen readers can only read text, how would they read the graphics to people who are blind? As soon as you start asking these types of questions, you begin to see that there are a few potential glitches in the accessibility of the Internet to people with disabilities. The Internet has the potential to revolutionize disability access to information, but if we're not careful, we can place obstacles along the way that destroy that potential, and which leave people with disabilities just as discouraged and dependent upon others as before.

People with Disabilities on the Web

Though estimates vary, most studies find that about one fifth (20%) of the population has some kind of disability. Not all of these people have disabilities that make it difficult for them to access the Internet. For example, a person whose legs are paralyzed can still navigate a web site without any disability-related difficulty. Still, if

only half—or even a quarter—of these individuals have disabilities that affect their ability to access the Internet, this is a significant portion of the population. Businesses would be unwise to purposely exclude 20, 10 or even 5 percent of their potential customers from their Web sites. Schools, universities, and government entities would be not only unwise, but, in many countries, they would also be breaking the law if they did so.

Each of the major categories of disabilities require certain types of adaptations in the design of the web content. Most of the time, these adaptations benefit nearly everyone, not just people with disabilities. For example, people with cognitive disabilities benefit from illustrations and graphics, as well as from properly-organized content with headings,

© Copyright Virtual University of Pakistan

lists, and visual cues in the navigation. Similarly, though captioned video content is meant to benefit people who are deaf, it can also benefit those who do not have sound on their computers, or who do not want to turn the sound on in public places such as libraries, airplanes, or computer labs.

Occasionally, Web developers must implement accommodations that are more specific to people with disabilities. For example, developers can add links that allow blind users or people with motor disabilities who cannot use a mouse to skip past the navigational links at the top of the page. People without disabilities may choose to use this feature as well, but they will usually ignore it. In almost every case, even these disability-specific adaptations can be integrated into the site's design with little or no impact to its overall visual "look and feel." Unfortunately, too many web developers are convinced that the opposite is true. They worry that their sites will become less appealing to their larger audience of people without disabilities. This faulty perception has led to countless circular debates, that tend to cause unnecessary friction between web designers and people with disabilities.

From the perspective of people with disabilities, inaccessible web content is an obstacle that prevents them from participating fully in the information revolution that has begun unfolding on the Internet. To them, it is a matter of basic human rights. When web developers truly understand this perspective, most of them realize the importance of the issue, and are willing to do what they can to make their Web content more accessible.

Comprehensive Solutions

There are two key components to any effort to achieve web accessibility:

- Commitment and accountability
- Training and technical support

Either of these by itself is insufficient.

Commitment and accountability

Awareness. The foundation of any kind of commitment to web accessibility is awareness of the issues. Most Web developers are not personally opposed to the concept of making the Internet accessible to people with disabilities. In fact, most accessibility errors on web sites are the result of ignorance, rather than malice or apathy. A large proportion of developers have simply never even thought about the issue. Even if they have heard of web accessibility, they may not understand what's at stake. Their ignorance leads them to ask questions such as, "Why would a blind person want to access the Internet?" After hearing an explanation of the ways in which blind people can access the Internet and the reasons why they have difficulties with some sites, most of these same developers understand the importance of the issue, and most are willing to do something about it, at least in the abstract.

Leadership. Understanding the issues is an important first step, but it does not solve the problem, especially in large organizations. If the leadership of an organization does not express commitment to web accessibility, chances are low that the organization's web content will be accessible. Oftentimes, a handful of developers make their own content accessible while the majority don't bother to, since it is not expected of them.

Policies and Procedures. Even when leaders express their commitment to an idea, if the idea is not backed up by policies, the idea tends to get lost among the day-to-day routines. The best approach for a large organization is to create an internal policy that outlines

© Copyright Virtual University of Pakistan

specific standards, procedures, and methods for monitoring compliance with the standards and procedures. For example, an organization's policy could be that Web developers will create content that complies with the web Content Accessibility Guidelines of the W3C, that no content is allowed to go live on the web site until it has been verified to meet this standard, and that the site will be re-examined quarterly for accessibility errors. This example won't fit every situation or every organization, but it does at least provide a simplified theoretical model from which to create standards, procedures, and monitoring methods within organizations.

Training and technical support

Sometimes web developers fear that it is more expensive and time-consuming to create accessible web sites than it is to create inaccessible ones. This fear is largely untrue. On a page-by-page basis, the extra time required by a knowledgeable developer to make the content accessible is so minimal as to be almost negligible. Once developers know the concepts, implementing them becomes second-nature, and does not add significantly to the total development time.

However, it does take time to become a knowledgeable developer. A developer can learn the basics of Web accessibility in just a few days, but, as with any technical skill, it often takes months to internalize the mindset as well as the techniques. Organizations should ensure that their developers have access to training materials, workshops, books, or courses which explain the details of accessible web design.

Some of these resources are available for free, such as the WebAIM web site.

However, not everyone learns best in an online environment. Sometimes the best approach is to invite an outside consultant to provide training through presentations, workshops, or one-on-one tutoring.

Ongoing technical support can be offered through outside consultants, discussion groups, internal workshops, classes or other methods. Some organizations have set up their own internal discussion groups to provide a forum for talking about accessibility issues. If a developers forum already exists at an organization, it may be unnecessary to create a new one specifically for accessibility if the existing one can serve the same purpose. The WebAIM forum consists of people from all over the world who are interested in Web accessibility issues, many of whom are highly knowledgeable about the topic and willing to share their knowledge with others.

Conclusion

The web offers so many new opportunities to people with disabilities that are unavailable through any other medium. It provides a method for accessing information, making purchases, communicating with the world, and accessing entertainment that does not depend on the responsiveness of other people. The Internet offers independence and freedom. But this independence and freedom is only partially a reality. Too many web sites are not created with web accessibility in mind. Whether purposefully or not, they exclude the segment of the population that in many ways stands to gain the most from the Internet. Only by committing to accessibility and providing for accountability, training, and technical assistance, can the web's full potential for people with disabilities become a reality.

© Copyright Virtual University of Pakistan

Lecture 45.

Conclusion

Learning Goals

As the aim of this lecture is to introduce you the study of Human Computer Interaction, so that after studying this you will be able to:

- Understand the new and emerging interaction paradigms

The most common interaction paradigm in use today is the desktop computing paradigm. However, there are many other different types of interaction paradigms. Some of these are given below and many are still emerging:

Ubiquitous computing

Ubiquitous computing (ubicomp, or sometimes ubiqcomp) integrates computation into the environment, rather than having computers which are distinct objects. Other terms for ubiquitous computing include pervasive computing, calm technology, and things that think. Promoters of this idea hope that embedding computation into the environment and everyday objects would enable people to move around and interact with information and computing more naturally and casually than they currently do. One of the goals of ubiquitous computing is to enable devices to sense changes in their environment and to automatically adapt and act based on these changes based on user needs and preferences.

The late Mark Weiser wrote what are considered some of the seminal papers in Ubiquitous Computing beginning in 1988 at the Xerox Palo Alto Research Center (PARC). Weiser was influenced in a small way by the dystopian Philip K. Dick novel Ubik, which envisioned a future in which everything -- from doorknobs to toilet-paper holders, were intelligent and connected. Currently, the art is not as mature as Weiser hoped, but a considerable amount of development is taking place.

The MIT Media Lab has also carried on significant research in this field, which they call Things That Think.

American writer Adam Greenfield coined the term Everyware to describe technologies of ubiquitous computing, pervasive computing, ambient informatics and tangible media. The article All watched over by machines of loving grace contains the first use of the term. Greenfield also used the term as the title of his book Everyware: The Dawning Age of Ubiquitous Computing (ISBN 0321384016).

Early work in Ubiquitous Computing

The initial incarnation of ubiquitous computing was in the form of "tabs", "pads", and "boards" built at Xerox PARC, 1988-1994. Several papers describe this work, and there are web pages for the Tabs and for the Boards (which are a commercial product now):

Ubicomp helped kick off the recent boom in mobile computing research, although it is not the same thing as mobile computing, nor a superset nor a subset.

Ubiquitous Computing has roots in many aspects of computing. In its current form, it was first articulated by Mark Weiser in 1988 at the Computer Science Lab at Xerox PARC. He describes it like this:

Ubiquitous Computing #1

© Copyright Virtual University of Pakistan

Inspired by the social scientists, philosophers, and anthropologists at PARC, we have been trying to take a radical look at what computing and networking ought to be like. We believe that people live through their practices and tacit knowledge so that the most powerful things are those that are effectively invisible in use. This is a challenge that affects all of computer science. Our preliminary approach: Activate the world. Provide hundreds of wireless computing devices per person per office, of all scales (from 1" displays to wall sized). This has required new work in operating systems, user interfaces, networks, wireless, displays, and many other areas. We call our work "ubiquitous computing". This is different from PDA's, dynabooks, or information at your fingertips. It is invisible, everywhere computing that does not live on a personal device of any sort, but is in the woodwork everywhere.

Ubiquitous Computing #2

For thirty years most interface design, and most computer design, has been headed down the path of the "dramatic" machine. Its highest ideal is to make a computer so exciting, so wonderful, so interesting, that we never want to be without it. A less-traveled path I call the "invisible"; its highest ideal is to make a computer so imbedded, so fitting, so natural, that we use it without even thinking about it. (I have also called this notion "Ubiquitous Computing", and have placed its origins in post-modernism.) I believe that in the next twenty years the second path will come to dominate. But this will not be easy; very little of our current systems infrastructure will survive. We have been building versions of the infrastructure-to-come at PARC for the past four years, in the form of inch-, foot-, and yard-sized computers we call Tabs, Pads, and Boards. Our prototypes have sometimes succeeded, but more often failed to be invisible. From what we have learned, we are now explorting some new directions for ubicomp, including the famous "dangling string" display.

45.1 Wearable Computing

Personal Computers have never quite lived up to their name. There is a limitation to the interaction between a user and a personal computer. Wearable computers break this boundary. As the name suggests these computers are worn on the body like a piece of clothing. Wearable computers have been applied to areas such as behavioral modeling, health monitoring systems, information technologies and media development. Government organizations, military, and health professionals have all incorporated wearable computers into their daily operations. Wearable computers are especially useful for applications that require computational support while the user's hands, voice, eyes or attention are actively engaged with the physical environment.

Wristwatch videoconferencing system running GNU Linux, later featured in Linux Journal and presented at ISSCC2000One of the main features of a wearable computer is constancy. There is a constant interaction between the computer and user, ie. there is no need to turn the device on or off. Another feature is the ability to multi-task. It is not necessary to stop what you are doing to use the device; it is augmented into all other actions. These devices can be incorporated by the user to act like a prosthetic. It can therefore be an extension of the user’s mind and/or body.

Such devices look far different from the traditional cyborg image of wearable computers, but in fact these devices are becoming more powerful and more wearable all the time. The most extensive military program in the wearables arena is the US Army's Land Warrior system, which will eventually be merged into the Future Force Warrior system.

© Copyright Virtual University of Pakistan

Issues

Since the beginning of time man has fought man. The difference between the 18th century and the 21st century however, is that we are no longer fighting with guns but instead with information. One of the most powerful devices in the past few decades is the computer and the ability to use the information capabilities of such a device have transformed it into a weapon.

Wearable computers have led to an increase in micro-management. That is, a society characterized by total surveillance and a greater influence of media and technologies. Surveillance has impacted more personal aspects of our daily lives and has been used to punish civilians for seemingly petty crimes. There is a concern that this increased used of cameras has affected more personal and private moments in our lives as a form of social control.

History

Depending on how broadly one defines both wearable and computer, the first wearable computer could be as early as the 1500s with the invention of the pocket watch or even the 1200s with the invention of eyeglasses. The first device that would fit the modern-day image of a wearable computer was constructed in 1961 by the mathematician Edward O. Thorp, better known as the inventor of the theory of card-counting for blackjack, and Claude E. Shannon, who is best known as "the father of information theory." The system was a concealed cigarette-pack sized analog computer designed to predict roulette wheels. A data-taker would use microswitches hidden in his shoes to indicate the speed of the roulette wheel, and the computer would indicate an octant to bet on by sending musical tones via radio to a miniature speaker hidden in a collaborators ear canal. The system was successfully tested in Las Vegas in June 1961, but hardware issues with the speaker wires prevented them from using it beyond their test runs. Their wearable was kept secret until it was first mentioned in Thorp's book Beat the Dealer (revised ed.) in 1966 and later published in detail in 1969. The 1970s saw rise to similar roulette-prediction wearable computers using next-generation technology, in particular a group known as Eudaemonic Enterprises that used a CMOS 6502 microprocessor with 5K RAM to create a shoe-computer with inductive radio communications between a data-taker and better (Bass 1985).

In 1967, Hubert Upton developed an analogue wearable computer that included an eyeglass-mounted display to aid lip reading. Using high and low-pass filters, the system would determine if a spoken phoneme was a fricative, stop consonant, voiced-fricative, voiced stop consonant, or simply voiced. An LED mounted on ordinary

eyeglasses illuminated to indicate the phoneme type. The 1980s saw the rise of more general-purpose wearable computers. In 1981 Steve Mann designed and built a backpack-mounted 6502-based computer to control flash-bulbs, cameras and other photographic systems. Mann went on to be an early and active researcher in the wearables field, especially known for his 1994 creation of the Wearable Wireless Webcam (Mann 1997). In 1989 Reflection Technology marketed the Private Eye head-mounted display, which scanned a vertical array of LEDs across the visual field using a vibrating mirror. 1993 also saw Columbia University's augmented-reality system known as KARMA: Knowledge-based Augmented Reality for Maintenance Assistance. Users would wear a Private Eye display over one eye, giving an overlay effect when the real world was viewed with both eyes open. KARMA would overlay wireframe schematics and maintenance instructions on top of whatever was being repaired. For example, graphical wireframes on top of a laser printer would explain how to change the paper tray. The system used sensors attached to objects in the physical world to determine their locations, and the entire system ran tethered from a desktop computer (Feiner 1993).

© Copyright Virtual University of Pakistan

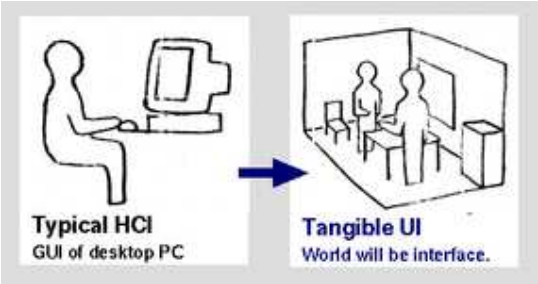
The commercialization of general-purpose wearable computers, as led by companies such as Xybernaut, CDI and ViA Inc, has thus far met with limited success. Publicly-traded Xybernaut tried forging alliances with companies such as IBM and Sony in order to make wearable computing widely available, but in 2005 their stock was delisted and the company filed for Chapter 11 bankruptcy protection amid financial scandal and federal investigation. In 1998 Seiko marketed the Ruputer, a computer in a (fairly large) wristwatch, to mediocre returns. In 2001 IBM developed and publicly displayed two prototypes for a wristwatch computer running Linux, but the product never came to market. In 2002 Fossil, Inc. announced the Fossil WristPDA, which ran the Palm OS. Its release date was set for summer of 2003, but was delayed several times and was finally made available on January 5, 2005.

45.2 Tangible Bits

The development from desktop to physical environment can be divided into two phases: the first one was introduced by the Xerox Star workstation in 1981. This workstation was the first generation of a graphical user interface that sets up a "desktop metaphor". It simulates a real physical desktop on a computer screen with a mouse, windows and icons. The Xerox Star workstation also establishes some important HCI design principles like "seeing and pointing".

Ten years later, in 1991, Marc Weiser illustrates a different paradigm of computing, called "ubiquitous computing". His vision contains the displacement of computers into the background and the attempt to make them invisible.

A new paradigm desires to start the next period of computing by establish a new type of HCI called "Tangible User Interfaces" (TUIs). Herewith they try to make computing truly ubiquitous and invisible. TUIs will change the world itself to an interface (see figure below) with the intention that all surfaces (walls, ceilings, doors…) and objects in the room will be an interface between the user and his environment.



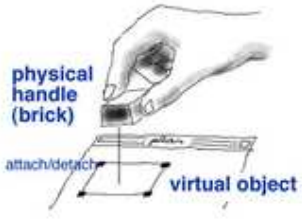
Below are some examples:

The ClearBoard (TMG, 1990-95) has the idea of changing a passive wall to an active dynamic collaboration medium. This leads to the vision, that all surfaces become active surfaces through which people can interact with other (real and virtual) spaces (see figure below).

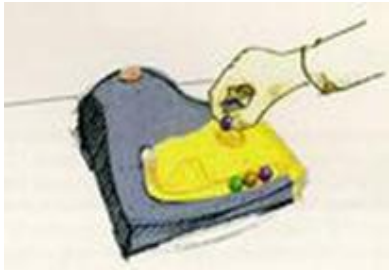
© Copyright Virtual University of Pakistan



Bricks (TMG, 1990-95) is a graphical user interface that allows direct control of virtual objects through handles called "Bricks". These Bricks can be attached to virtual objects and thus make them graspable. This project encouraged two-handed direct manipulation of physical objects (see figure below).



The Marble Answering Machine (by Durell Bishop, student at the Royal College of Art) is a prototype telephone answering machine. Incoming voice messages are represented by marbles, the user can grasp and then drop to play the message or dial the caller automatically. It shows that computing doesn't have to take place at a desk, but it can be integrated into everyday objects. The Marble Answering Machine demonstrates the great potential of making digital information graspable (see figure below).



Goals and Concepts of "tangible bits"

In our world there exist two realms: the physical environment of atoms and the cyberspace of bits. Interactions between these two spheres are mostly restricted to GUI-based boxes and as a result separated from ordinary physical environments. All senses, work practices and skills for processing information we have developed in the past are often neglected by our current HCI designs.

Goals

So there is a need to augment the real physical world by coupling digital information to everyday things. In this way, they bring the two worlds, cyberspace and real world, together by making digital information tangible. All states of physical matter, that means, not only solid matter, but also liquids and gases become interfaces between people and cyberspace. It intends to allow users both to "grasp and manipulate" foreground bits and be aware of background bits. They also don't want to have a distinction between special

© Copyright Virtual University of Pakistan

input and output devices any longer, e.g. between representation and control. Nowadays, interaction devices are divided into input devices like mice or keyboards and output devices like screens. Another goal is not to have a one-to-one mapping between physical objects and digital information, but an aggregation of several digital information instead.

Concepts

To achieve these goals, they worked out three key concepts: “interactive surfaces”, “coupling bits and atoms” and “ambient media”. The concept “interactive surfaces” suggests a transformation of each surface (walls, ceilings, doors, desktops) into an active interface between physical and virtual world. The concept “coupling bits and atoms” stands for the seamless coupling of everyday objects (card, books, and toys) with digital information. The concept “ambient media” implies the use of sound, light, air flow, water movement for background interfaces at the periphery of human perception.

45.3 Attentive Environments

Attentive environments are environments that are user and context aware. One project which explores these themes is IBM's BlueEyes research project is chartered to explore and define attentive environments.

Animal survival depends on highly developed sensory abilities. Likewise, human cognition depends on highly developed abilities to perceive, integrate, and interpret visual, auditory, and touch information. Without a doubt, computers would be much more powerful if they had even a small fraction of the perceptual ability of animals or humans. Adding such perceptual abilities to computers would enable computers and humans to work together more as partners. Toward this end, the BlueEyes project aims at creating computational devices with the sort of perceptual abilities that people take for granted.

How can we make computers "see" and "feel"?

BlueEyes uses sensing technology to identify a user's actions and to extract key information. This information is then analyzed to determine the user's physical, emotional, or informational state, which in turn can be used to help make the user more productive by performing expected actions or by providing expected information. For example, a BlueEyes-enabled television could become active when the user makes eye contact, at which point the user could then tell the television to "turn on".

In the future, ordinary household devices -- such as televisions, refrigerators, and ovens -- may be able to do their jobs when we look at them and speak to them.

© Copyright Virtual University of Pakistan