A risk analysis involves quantifying degree of uncertainty of the risk and loss associate with it. In this regards, the PM tries to find answers to the following questions:

- What can go wrong?
- What is the likelihood of it going wrong?
- What will the damage be?
- What can we do about it?

Masters
highlighted by Masters

Final Term Highlighted Handouts by Masters

Lecture No. 19

Types of Risks

Each project is faced with many types of risks. These include:

- Project risks
  - Will impact schedule and cost
  - Includes budgetary, schedule, personnel, resource, customer, requirement problems
- Technical risks
  - Impact the quality, timelines, and cost
  - Implementation may become difficult or impossible
  - Includes design, implementation, interface, verification and maintenance problems
  - Leading edge technology
- Business risks
  - Marketability
  - Alignment with the overall business strategy
  - How to sell
  - Losing budget or personnel commitments

Furthermore, there are predictable and unpredictable risks. Predictable risks can be uncovered after careful evaluation whereas unpredictable risks cannot be identified.

Risk Identification

It is the responsibility of the project manager to identify known and predictable risks. These risks fall in the following categories of generic risks and product specific risks. *Generic risks* are threats to every project whereas *Product specific risks* are specific to a particular project. The question to be asked in this context is: what special characteristics of this project may threaten your project plan? A useful technique in this regards is the preparation of a risk item checklist. This list tries to ask and answer questions relevant to each of the following topics for each software project:

- Product size
- Business impact
- Customer characteristics
- Process definition
- Development environment
- Technology to be built
- Staff size and experience

Assessing Overall Project Risks

In order to assess the overall project risks, the following questions need to be addressed:

- Have top software and customer managers formally committed to support the project?
- Are end-users committed to the project and the system/product to be built?
- Are requirements fully understood?
- Have customers been involved fully in requirement definition?
- Do end-users have realistic expectations?
- Does the software team have right mix of skills?

© Copy Right Virtual University of Pakistan

- Are project requirements stable?
- Does the project team have experience with the technology to be implemented?
- Is the number of people on the project team adequate to do the job?

Risk components and drivers

Each risk has many components and forces behind them. From this perspective, risks can be categorized into the following categories:

- Performance risks
  - Degree of uncertainty that the product will meet its requirements and be fit for its intended use
- Cost risks
  - The degree of uncertainty that the project budget will be maintained
- Support risks
  - Resultant software will be easy to correct, enhance, and adapt
- Schedule risks
  - Product schedule will be maintained

Each risk has its own impact and can be characterized as negligible, marginal, critical, or catastrophic.

Marginal minor and not important

This is summarized in the following table:

Risk Impact		Performance	Support	Cost	Schedule
Catastrophic	Consequence of error	Failure to meet the requirements will result in mission failure		Results in increased cost and schedule delays. Expected value in excess of $500K	
	Consequence of failure to achieve desired result	Significant degradation	Non-responsive or unsupportable	Budget overrun likely	Unachievable
Critical	Consequence of error	Would degrade performance to a point where mission success is questionable		Results in operational delays and or increased cost with expected value of $100K-$500K	
	Consequence of failure to achieve desired result	Some reduction in technical performance	Minor delays	Possible overrun	Possible slippage
Marginal	Consequence of error	Result in degradation of secondary mission		Expected value <$100K	
	Consequence of failure to achieve desired result	Small reduction	Responsive	Sufficient financial resources	Realistic
Negligible	Consequence of error	Inconvenience		Minor	
	Consequence of failure to	No reduction	Supportable	Budget under run	achievable

Catastrophic risk—the chance that a large number of people will suffer a given level of harm.

	achieve desired result			possible	

Risk Projection

Risk projection is concerned with risk estimation. It attempts to rate risks in two ways: likelihood and consequences. There are four risk project activities. These are:

- Establish a scale that reflects the perceived likelihood of risk
- Delineate the consequences
- Estimate impact
- Note the overall accuracy of risk projection

This process is exemplified with the help of the following table:

Risk	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	

Where impacts are codified as follows:
1: catastrophic 2: critical 3: marginal 4: negligible

and RMMM stands for risk mitigation, monitoring, and management plan.

Lecture No. 20

Assessing Risk Impact

Assessment of risk impact is a non-trivial process. Factors affecting the consequences are the nature, scope, and timing.

For each risk an exposure is calculated as follows:

RE = Probability of the risk x Cost

This risk exposure is then used to identify the top risks and mitigation strategies.

As an example, let us consider the following case:

- Risk:
  - Only 70% of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.
- Risk Probability – 80% likely (i.e. 0.8)
- Risk impact
  - 60 reusable software components were planned. If only 70% can be used, 18 components would have to be developed from scratch. Since the average component will cost 100,000, the total cost will be 1,800,000.
- Therefore, RE = 0.8 * 1,800,000 = 1,440,000

A high level risk may be refined into finer granularity to handle it efficiently. As an example, the above mentioned risk is refined as follows:

1. Certain reusable components were developed by 3rd party with no knowledge of internal design standard.
2. Design standard for component interfaces has not been solidified and may not conform to certain existing components.
3. Certain reusable components have been implemented in a language that is not supported on the target environment.

We can now take the following measures to mitigate and monitor the risk:

1. Contact 3rd party to determine conformance with design standards.
2. Press for interface standard completion; consider component structure when deciding on interface protocol.
3. Check to determine if language support can be acquired.

This leads us to the following Management/Contingency Plan:

1. RE computed to 1,440,000. Allocate this amount within project contingency cost.
2. Develop revised schedule assuming 18 additional components will have to be custom-built
3. Allocate staff accordingly

Risk Mitigation, Monitoring, and Management (RMMM)

The RMMM plan assists the project team in developing strategy for dealing with risk. In this context, an effective strategy must consider:

- Risk avoidance
- Risk monitoring
- Risk management and contingency plan

It must always be remembered that avoidance is always the best strategy.

© Copy Right Virtual University of Pakistan

As an example, let consider the following scenario. In this case high turn-over has been identified as a risk with the following characteristics:

- Risk r_j - High turnover
- Likelihood $l_j = 0.7$
- Impact x_j - projected at level 2 (critical)

Let us now devise a mitigation strategy for reducing turnover. In order to do so, the following steps may be taken:

Risk mitigation focuses on minimizing the harm of a particular risk

- Meet with current staff to determine causes for turnover (e.g. poor working conditions, low pay, competitive job market)
- Mitigate those causes that are under our control before the project starts
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave
- Organize project teams so that information about each development activity is widely dispersed
- Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner (to ensure continuity)
- Conduct peer reviews of all work (so that more than one person is up to speed)
- Assign a backup staff member for every critical technology

Once the strategy has been devised, the project must be monitored for this particular risk. That is, we must keep an eye on the various factors that can indicate that this particular risk is about to happen. In this case, the factors could be:

- General attitude of team members based on project pressures
- The degree to which the team is jelled
- Interpersonal relationships among team members
- Potential problems with compensation and benefits
- The availability of jobs within the company and outside it

Also, the effectiveness of the risk mitigation steps should be monitored. So, in this example, the PM should monitor documents carefully to ensure that each can stand on its own and that each imparts information that would be necessary if a newcomer were forced to join the software team somewhere in the middle of the project.

Risk Management and Contingency Plan

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.

- Risk has become a reality – some people announce that they will be leaving
- If mitigation strategy has been followed, backup is available, information has been documented, and knowledge has been dispersed
- Temporarily refocus and readjust resources
- People who are leaving are asked to stop all work and ensure knowledge transfer

© Copy Right Virtual University of Pakistan

Risk mitigation and contingency is a costly business. It is therefore important to understand that for RMMM plan, a cost/benefit analysis of each risk must be carried out. The Pareto principle (80-20 rule) is applicable in this case as well – 20% of the identified risk account for 80% of the potential for project failure.

Masters
Subscribes to Masters

Masters
highlighted by Masters

Lecture No. 21

Software Project Scheduling and Monitoring

Software project scheduling is the next task to be performed by the PM. It is important to note once again that in the reasons for project failure, unrealistic deadline and underestimate of effort involved in the project are two of the most important reasons for project failure. Therefore, a good schedule estimate would increase the chances of the success of the project.

In this context, a PM has to first come up with the schedule and then monitor the progress of the project to ensure that things are happening according to the schedule. It would not be out of place to quote Fred Brooks at this point. He says, *"Projects fall behind schedule one day at a time."* That means a delay of a week or a month or a year does not happen suddenly – it happens one day at a time. Therefore, a project manager has to be vigilant to ensure that the project does not fall behind schedule.

The reality of a technical project is that hundreds of small tasks must occur to accomplish a large goal. Therefore the Project manager's objectives include:

- Identification and definition all project tasks
- Building a network that depicts their interdependencies
- Identification of the tasks that are critical within the network
- Tracking their progress to ensure delay is recognized one day at a time

For this, the schedule must be fine grained.

Software Project Scheduling

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

It is important to note that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major SE activities and the product functions to which they are applied. As the project gets underway these tasks are refined into a detailed schedule.

In order to come up with a realistic schedule, the following basic principles are used:

- Compartmentalization

The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and process are decomposed.

- Interdependency

The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some activities cannot commence until the work product produced by another is available.

© Copy Right Virtual University of Pakistan

- Time allocation

Each task to be scheduled must be allocated some number of work units (e.g. person-days of effort). In addition, each task must be assigned a start date and an end date which is a function of the interdependencies and number of resources.

- Effort validation

Every project has a defined number of staff members. As time allocation occurs, the project manager must ensure that no more than the allocated number of people has been scheduled at any given time.

- Defined responsibilities

Every task should be assigned to a specific team member.

- Defined outcomes

Every task should have a defined outcome, normally a work product.

- Defined milestones

Every task or group of tasks should be associated with a project milestone.

Software Project Scheduling and Monitoring

Software project scheduling is the next task to be performed by the PM. It is important to note once again that in the reasons for project failure, unrealistic deadline and underestimate of effort involved in the project are two of the most important reasons for project failure. Therefore, a good schedule estimate would increase the chances of the success of the project.

In this context, a PM has to first come up with the schedule and then monitor the progress of the project to ensure that things are happening according to the schedule. It would not be out of place to quote Fred Brooks at this point. He says, *"Projects fall behind schedule one day at a time."* That means a delay of a week or a month or a year does not happen suddenly – it happens one day at a time. Therefore, a project manager has to be vigilant to ensure that the project does not fall behind schedule.

The reality of a technical project is that hundreds of small tasks must occur to accomplish a large goal. Therefore the Project manager's objectives include:

– Identification and definition all project tasks
– Building a network that depicts their interdependencies
– Identification of the tasks that are critical within the network
– Tracking their progress to ensure delay is recognized one day at a time

For this, the schedule must be fine grained.

Software Project Scheduling

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.
It is important to note that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major

© Copy Right Virtual University of Pakistan

SE activities and the product functions to which they are applied. As the project gets underway these tasks are refined into a detailed schedule.

In order to come up with a realistic schedule, the following basic principles are used:

- Compartmentalization

The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and process are decomposed.

- Interdependency

The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some activities cannot commence until the work product produced by another is available.

- Time allocation

Each task to be scheduled must be allocated some number of work units (e.g. person-days of effort). In addition, each task must be assigned a start date and an end date which are a function of the interdependencies and number of resources.

- Effort validation

Every project has a defined number of staff members. As time allocation occurs, the project manager must ensure that no more than the allocated number of people have been scheduled at any given time.

- Defined responsibilities

Every task should be assigned to a specific team member.

- Defined outcomes

Every task should have a defined outcome, normally a work product.

- Defined milestones

Every task or group of tasks should be associated with a project milestone.

© Copy Right Virtual University of Pakistan

Lecture No. 22

Relationship between people and effort

The relationship between the number of people and time to develop an application is not linear. It is not as simple as a 120 man-day project can be developed by 1 person working for 120 days or 120 people working for 1 day. The communication and coordination overhead plays a very significant role.

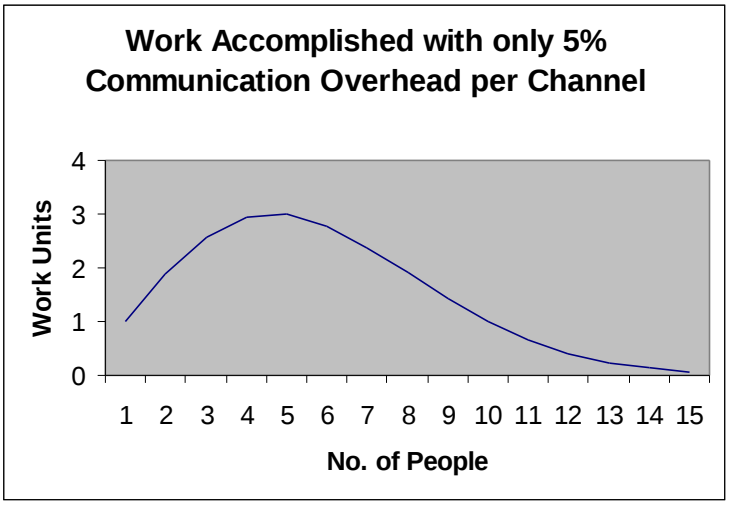
As can be recalled from our earlier discussions, total number of Channels of communication involving N people is given by the following formula.

$$C = N(N-1)/2$$

Now, if the communication overhead per channel is k, then work accomplished is given by:

$$W = (1-k)^C \times N$$

This phenomenon is depicted in the following diagram:



It may be noted here that with only a 5% communication overhead per channel, the total work accomplished by a team of 6 people would be less than the volume of work completed by a team of 4 people. It is also interesting to note that it approaches 0 as the team size approaches 20.

Task Set Definition

A process model defines a task set which comprises of SE work tasks, milestones, and deliverables. This enable a software team to define, develop, and support the software.

Therefore, each software process should define a collection of task sets, designed to meet the needs of different types of projects. To determine the set of tasks to be performed the type of the project and the degree of rigor required needs to be established. Different

© Copy Right Virtual University of Pakistan

types of projects and different degree of rigor. These projects could fall into the following categories:

- Concept development projects
- New application development
- Application enhancement
- Application maintenance
- Reengineering projects

The degree of rigor can also be categorized as Casual, Structured, Strict, or Quick Reaction. The following paragraphs elaborate each one of these.

- Casual
  All process framework activities are applied, but only a minimum task set is required. It requires reduced umbrella tasks and reduced documentation. Basic principles of SE are however still followed.
- Structured
  In this case a complete process framework is applied. Appropriate framework activities, related tasks, and umbrella activities (to ensure high quality) are also applied. SQA, SCM, documentation, and measurement are conducted in streamlined manner.
- Strict
  In this case a full process is implemented and all umbrella activities are applied. The work products generated in this case are robust.
- Quick Reaction
  This approach is taken in case of an emergency. In this case only task essential for maintaining good quality are applied. After the task has been accomplished, documents are updated by back-filling.

The next question is how to decide about the degree of rigor. For this purpose an adaptation criterion has been developed. The following parameters are considered before a decision is made:

- Size of the project
- Number of potential users
- Mission criticality
- Application longevity
- Stability of requirements
- Ease of customer/developer communication
- Size of the project
- Number of potential users
- Mission criticality
- Application longevity
- Stability of requirements
- Ease of customer/developer communication

These parameters are used to calculate what is known as **Task Set Selector (TSS)** which is then used to determine the degree of rigor. It is computed as follows:

© Copy Right Virtual University of Pakistan

Each adaptation criteria is assigned a grade, weight, and entry point multiplier.

1. A grade value of between 1 and 5 is assigned to each adaptation criteria as appropriate.
2. The default value of the weighting factor (as given in the table below) is reviewed and modify if needed. The ranges should be between 0.8 to 1.2.

Adaptation Criteria	Grade	Weight	Entry Point Multiplier					Product
			Conc.	New Dev.	Enhan.	Maint.	Reeng.	
Size of product		1.2	0	1	1	1	1	
Number of Users		1.1	0	1	1	1	1	
Business Criticality		1.1	0	1	1	1	1	
Longevity		0.9	0	1	1	0	0	
Stability of requirements		1.2	0	1	1	1	1	
Ease of communication		0.9	0	1	1	1	1	
Maturity of technology		0.9	1	1	0	0	1	
Performance Constraints		0.8	0	1	1	0	1	
Embedded / non-embedded		1.2	1	1	1	0	1	
Project Staffing		1.0	1	1	1	1	1	
Interoperability		1.1	0	1	1	1	1	
Reengineering factors		1.2	0	0	0	0	1	
Task set selector								

3. Multiply the grade entered by the weight and by entry point multiplier for the type of project undertaken. The entry point multiplier takes a value of 0 or 1 and indicates the relevance of the adaptation criterion to the project type.

Product = grade × weight × entry point multiplier

The TSS is then computed as the average of all the entries in the product column. It is then used to determine the degree of rigor required as shown in the table below.

TSS	Degree of Rigor
< 1.2	Casual
1.0 – 3.0	Structured
> 2.4	Strict

It may be noted that there is overlap in TSS values. This overlap is intentional and it depicts that sharp boundaries are impossible to define when making task set selection. In the final analysis, the TSS value, past experience, and common sense must all be factored into the choice of the task set for a project.

© Copy Right Virtual University of Pakistan

This concept is demonstrated with the help of following example. In this case TSS value is computed for a new development project.

Adaptation Criteria	Grade	Weight	Entry point multiplier	Product
			New Development	
Size of product	2	1.2	1	2.4
Number of Users	3	1.1	1	3.3
Business Criticality	4	1.1	1	4.4
Longevity	3	0.9	1	2.7
Stability of requirements	2	1.2	1	2.4
Ease of communication	2	0.9	1	1.8
Maturity of technology	2	0.9	1	1.8
Performance Constraints	3	0.8	1	2.4
Embedded / non-embedded	3	1.2	1	3.6
Project Staffing	2	1.0	1	2.0
Interoperability	4	1.1	1	4.4
Reengineering factors	0	1.2	0	0
TSS				2.6

Since the value of TSS is 2.6, the degree of rigor could be structured or strict. The project manager now needs to use his experience to determine which of the two approaches should be taken.

TSS	**Degree of Rigor**
< 1.2	Casual
1.0 – 3.0	Structured
> 2.4	Strict

Example – SE Tasks for a Concept Development Project

Masters
Subscribes to Masters

Lecture No. 23

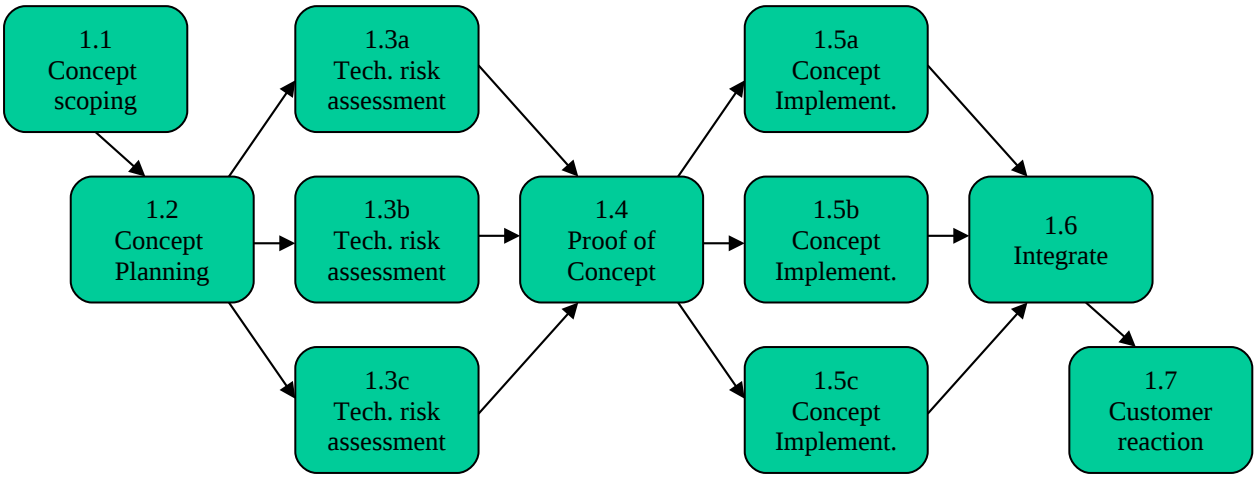
Task Network

Let us now apply the principles studied above to develop a plan for a concept development project. In this case the potential for new technology or new business idea is explored. Since it is a concept development project, the applicability is not certain but it appears to be useful and hence needs to be explored. Major tasks include:

- Concept scoping
- Preliminary concept planning
- Technology risk assessment
- Proof of concept
- Concept implementation
- Customer reaction to concept

Defining a Task Network

Once the tasks have been identified, we need to develop a task network to determine the sequence in which these activities need to be performed. This will ultimately lead to the time required to complete the project (to be discussed later). The following diagram shows the task network for the above project.



© Copy Right Virtual University of Pakistan

Lecture No. 24

Scheduling

Once we have the task network, we are now ready to prepare a schedule for the project. For this we use two techniques known as:

- Program evaluation and review techniques (PERT)
- Critical Path Method (CPM)

These are quantitative tools that allow the software planner to determine the critical path – the chain of tasks that determines the duration of the project and establish most likely time estimates for individual tasks by applying statistical models. They also help the planner to calculate boundary times that define a time window for a particular task.

The boundary time defines the following parameters for a project:

- The earliest time that a task can begin when all preceding tasks are completed in the shortest possible time
- The latest time for task initiation before the minimum project completion time is delayed
- The earliest finish
- The latest finish
- The total float – the amount of surplus time or leeway allowed in scheduling tasks so that the network critical path is maintained on schedule

In order to use the PERT and CPM, the following is required:

- A decomposition of product function
- A selection of appropriate process model and task set
- Decomposition of tasks – also known as the work breakdown structure (WBS)
- Estimation of effort
- Interdependencies

Timeline Chart

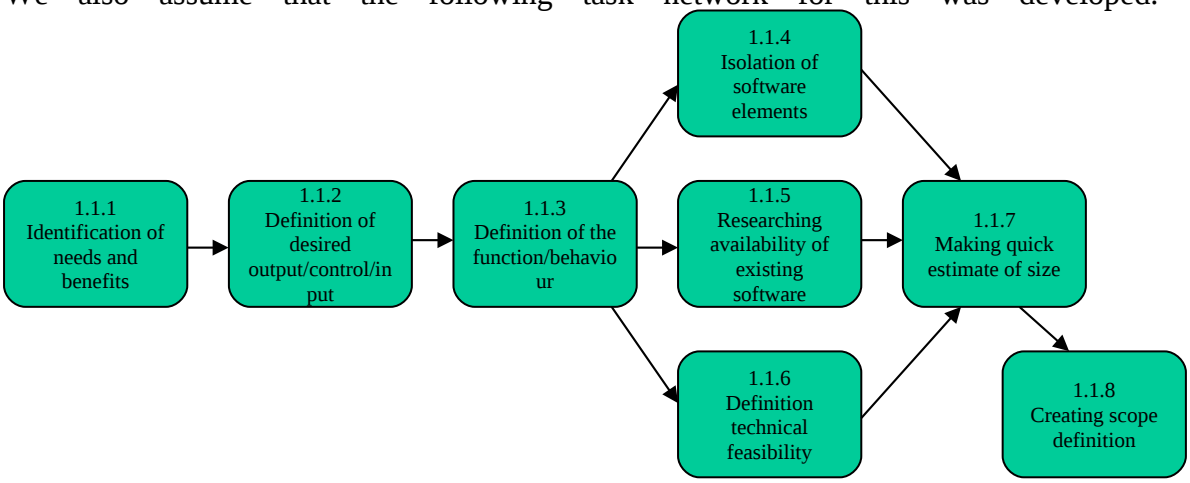
To develop the schedule for a project, time required for each activity in the Task Network is estimated. This analysis and decomposition leads to the development of a Timeline or Gantt Chart for the project which portrays the schedule for the project. As an example, let us assume that Concept Scoping (the first task in the above list) is further subdivided into the following sub-tasks with the associated estimated time requirements:

1. Identification of needs and benefits (3 days)
2. Definition of desired output/control/input (7 days)
3. Definition of the function/behaviour (6 days)
4. Isolation of software elements (1 day)
5. Researching availability of existing software (2 days)
6. Definition technical feasibility (4 days)
7. Making quick estimate of size (1 day)

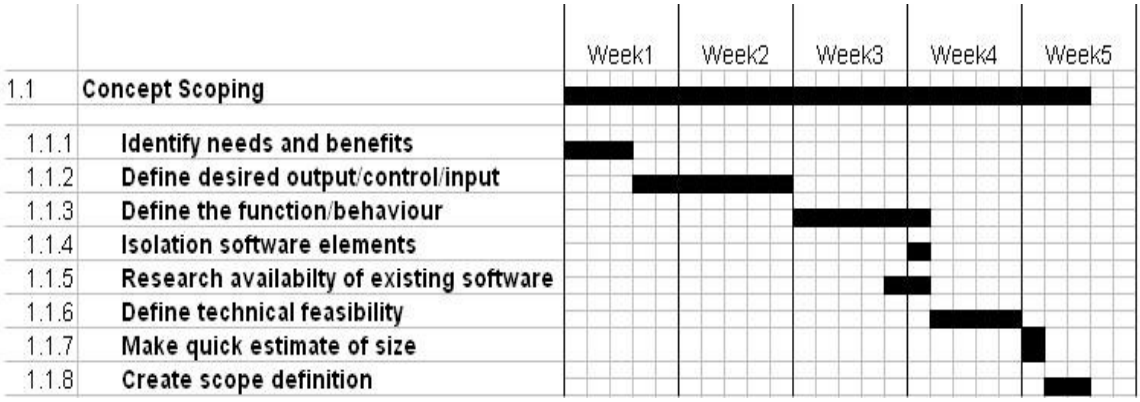
© Copy Right Virtual University of Pakistan

8. Creating scope definition (2 days)

We also assume that the following task network for this was developed.



This is now converted in the following schedule in the form of a Gantt Chart. Note that, the concept of boundary time allows us to schedule Task Numbers 1.1.4 and 1.1.5 anywhere along Task Number 1.1.3. The actual time is determined by the project manager is based upon the availability of resources and other constraints. Each task is further subdivided in sub-tasks in the same manner until the schedule for the complete project is determined.



Tracking a Schedule

Lecture No. 25

Project Tracking

A schedule is meaningless if it is not followed and tracked. Tasks and milestones defined in a project schedule must be tracked and controlled as project proceeds. Tracking methods include:

- Periodic project status meetings
- Evaluating the results of all reviews
- Determine whether project milestones have been accomplished by the scheduled date
- Comparing actual start date to planned start date
- Informal meetings with the practitioners
- Using earned value analysis
- Error tracking

The last two techniques are discussed in further detail in the following paragraphs:

Earned Value Analysis

Earned Value Analysis or EVA is a quantitative technique for assessing the progress of a project. The earned value system provides a common value scale for every software task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on the estimated percentage of the total. In order to do the EVA, the budgeted cost of work schedule (BCWS) is determined as follows:

Let

$BCWS_i$ = effort (person-days etc) for task *i*

BCWS is then the Progress so far – add all BCWS*i* so far.

Now

BAC = budget at completion = $\sum$ BCWS*i*

Now if BCWP is the Budgeted Cost of Work Perform, then

Schedule performance index	SPI = BCWP/BCWS
Schedule variance	SV = BCWP – BCWS

SPI close to 1 indicates efficient execution.

Similarly

Percent scheduled for completion = BCWS/BAC
Percent complete = BCWP/BAC
Actual cost work performed ACWP
Cost performance index CPI = BCWP/ACWP
Cost variance CV = BCWP – ACWP

Now, value of CPI close to 1 means project is within its defined budget.

Therefore, by using SPI and CPI we estimate how the project is progressing. If we have these values close to 1, it means that we have had good estimates and the project is under control.

© Copy Right Virtual University of Pakistan

Lecture No. 26

Error Tracking

Error tracking can also be used to estimate the progress of the project. In this case we track errors in work products (requirement specifications, design documents, source code etc) to assess the status of a project. The process works as follows:

We collect error related metrics over many projects and determine our defect removal efficiency in the following manner:

Defect removal efficiency, DRE = E / (E+D), where

- E – errors found before shipment
- D – errors found during operation

It provides a strong indication of the effectiveness of the quality assurance activities.

Now let us assume that we have collected the following errors and defect data over the last 24 months:

- Errors per requirement specification page – E_{req}
- Error per component – design level – E_{design}
- Errors per component – code level – E_{code}
- DRE – requirement analysis
- DRE – architectural design
- DRE – coding

We now record the number of errors found during each SE step and calculate current values for E_{req}, E_{design}, and $E_{code.}$ These values are then compared to averages of past projects. If the current results vary more than 20% from average, there may be cause for concern and there is certainly cause for investigation.

Example

- E_{req} for the current project = 2.1
- Organizational average = 3.6
  - Two possibilities
    - The team has done an outstanding job
    - The team has been lax in its review approach
  - If the second scenario appears likely
    - Build additional design time

This can also be used to better target review and/or testing resources in the following manner:

- 120 components
- 32 exhibit E_{design} > 1.2 average
- Adjust code review resources accordingly

Time Boxing

© Copy Right Virtual University of Pakistan

Time-boxing is used in severe deadline pressure. It is a use incremental strategy where tasks associated with each increment are time-boxed in the following manner:

- Schedule for each task is adjusted by working backward from the delivery date.
- A box is put around each task
- When a task hits the boundary of the box, work stops and next task begins

The principle behind time-boxing is the 90-10 rule (similar to Pareto Principle) – rather than becoming stuck on the 10% of a task, the product proceeds towards the delivery date in 90% of the cases.

© Copy Right Virtual University of Pakistan

Lecture No. 27

Software Quality Assurance

Quality cannot be assured without first understanding its nature and characteristics. So the first question one has to ask is: what is quality?

Software quality is defined as conformance to explicitly stated functional and non-functional requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

This definition emphasizes upon three important points:

- Software requirements are the foundation from which quality is measured. Lack of conformance is lack of quality
- Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
- A set of implicit requirements often goes unmentioned (ease of use, good maintainability etc.)

Another very important question is: Do you need to worry about it after the code has been generated? In fact, SQA is an umbrella activity that is applied throughout the software process.

Also, do we care about internal quality or the external quality? And finally, is there a relationship between internal and external qualities? That is, does internal quality translate in external quality?

In the literature, quality has been defined through in many different manners. One group believes that the quality has measurable characteristic such as cyclomatic complexity, cohesion, and coupling.
We can then talk about quality from different aspects. Quality of design tries to determine the quality of design related documents including requirements, specifications, and design. Quality of conformance looks at the implementation and if it follows the design then the resulting system meets its goals then conformance quality is high.
Are there any other issues that need to be considered? Glass defines quality as a measure of user satisfaction which is defined by

compliant product + good quality + delivery within budget and schedule

DeMarco defines product quality as a function of how much it changes the world for the better.

So, there are many different way to look at the quality.

Quality Assurance
Goal of quality assurance is to provide the management with the necessary data to be informed about product quality. It consists of auditing and reporting functions of

© Copy Right Virtual University of Pakistan

management. If data provided through QA identifies problems, the management deploys the necessary resources to fix it and hence achieves desired quality control.

Cost of quality

A very significant question is: does quality assurance add any value. That is, is worth spending a lot of money in quality assurance practices? In order to understand the impact of quality assurance practices, we have to understand the cost of quality (or lack thereof) in a system.

Quality has a direct and indirect cost in the form of cost of prevention, appraisal, and failure.

If we try to prevent problems, obviously we will have to incur cost. This cost includes:

- Quality planning
- Formal technical reviews
- Test equipment
- Training

We will discuss these in more detail in the later sections.

The cost of appraisal includes activities to gain insight into the product condition. It involves in-process and inter-process inspection and testing.

And finally, failure cost. Failure cost has two components: internal failure cost and external failure cost. Internal failure cost requires rework, repair, and failure mode analysis. On the other hand, external failure cost involves cost for complaint resolution, product return and replacement, help-line support, warranty work, and law suits.

It is trivial to see that cost increases as we go from prevention to detection to internal failure to external failure. This is demonstrated with the help of the following example:

Let us assume that a total of 7053 hours were spent inspecting 200,000 lines of code with the result that 3112 potential defects were prevented. Assuming a programmer cost of $40 per hour, the total cost of preventing 3112 defects was $382,120, or roughly $91 per defect.

Let us now compare these numbers to the cost of defect removal once the product has been shipped to the customer. Suppose that there had been no inspections, and the programmers had been extra careful and only one defect one 1000 lines escaped into the product shipment. That would mean that 200 defects would still have to be fixed in the field. As an estimated cost of $25000 per fix, the cost would be $5 Million or approximately 18 times more expensive than the total cost of defect prevention

That means, quality translates to cost savings and an improved bottom line.

SQA Activities

There are two different groups involved in SQA related activities:

- Software engineers who do the technical work
- SQA group who is responsible for QA planning, oversight, record keeping, analysis, and reporting

Software engineers address quality by applying solid technical methods and measures, conducting formal and technical reviews, and performing well planned software testing. The SQA group assists the software team in achieving a high quality product.

© Copy Right Virtual University of Pakistan

SQA Group Activities

An SQA plan is developed for the project during project planning and is reviewed by all stake holders. The plan includes the identification of:

- Evaluations to be performed
- Audits and reviewed to be performed
- Standards that are applicable to the project
- Procedures for error reporting and tracking
- Documents to be produced by the SQA group
- Amount of feedback provided to the software project team

The group participates in the development of the project's software process description. The software team selects the process and SQA group reviews the process description for compliance with the organizational policies, internal software standards, externally imposed standards, and other parts of the software project plan.
The SQA group also reviews software engineering activities to verify compliance with the defined software process. It identifies, documents, and tracks deviations from the process and verifies that the corrections have been made. In addition, it audits designated software work products to verify compliance with those defined as part of the software process. It, reviews selected work products, identifies, documents, and tracks deviations; verifies that corrections have been made; and reports the results of its work to the project manager.
The basis purpose is to ensure that deviations in software work and work products are documented and handled according to documented procedures. These deviations may be encountered in the project plan, process description, applicable standards, or technical work products. The group records any non-compliance and reports to senior management and non-compliant items are recorded and tracked until they are resolved.
Another very important role of the group is to coordinate the control and management of change and help to collect and analyze software metrics.

Quality Control

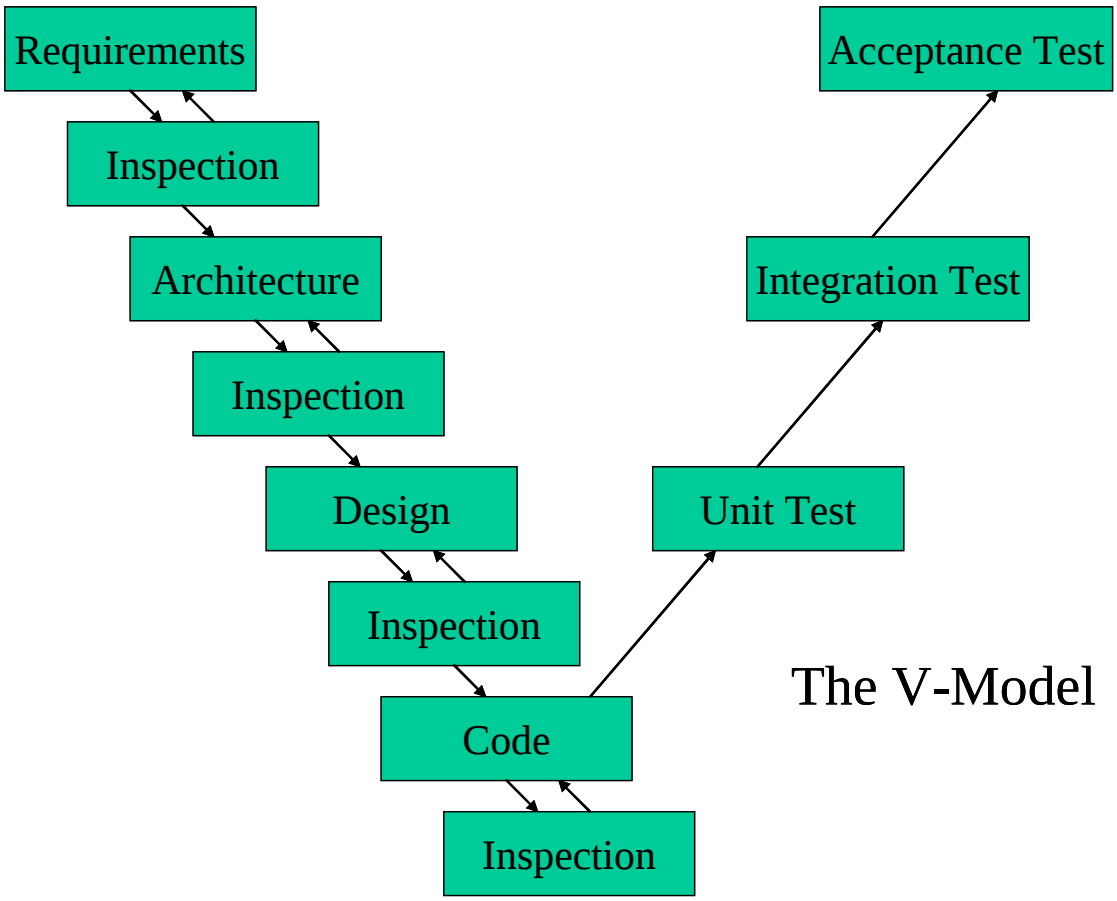
The next question that we need to ask is, once we have defined how to assess quality, how are we going to make sure that our processes deliver the product with the desired quality. That is, how are we going to control the quality of the product?
The basic principle of quality control is to control the variation as variation control is the heart of quality control. It includes resource and time estimation, test coverage, variation in number of bugs, and variation in support.
From one project to another we want to minimize the predicted resources needed to complete a project and calendar time. This involves a series of inspection, reviews, and tests and includes feedback loop. So quality control is a combination of measurement and feedback and combination of automated tools and manual interaction.

© Copy Right Virtual University of Pakistan

Lecture No. 28

Software Reviews

Software reviews are the filter for the software engineering process. They re applied at various different points and serve to uncover errors that can be removed and help to purify the software engineering activities.
In this context it is useful to look at the "V-model" of software development. This model emphasizes that SQA is a function performed at all stages of software development life cycle. At the initial stages (requirement, architecture, design, code), it is achieved through activities known as Formal Technical Reviews or FTR. At the later stages (integration and acceptance), testing comes into picture.



Importance of reviews

Technical work needs reviewing for the same reason that pencils needs erasers: To err is human. The second reason that we need technical reviews is although that people are good at catching errors, large class of errors escape the originator more easily than they escape anyone else.

Freedman defines a review – any review – as a way of using the diversity of a group of people to:

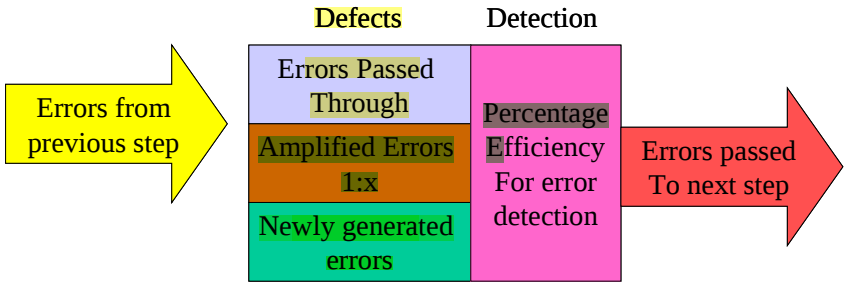
- Point out needed improvements in the product of a single person or team
- Confirm those parts of a product in which improvement is either not desired or no needed
- Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

Reviews help the development team in improving the defect removal efficiency and hence play an important role in the development of a high-quality product.

Types of Reviews

There are many types of reviews. In general they can be categorized into two main categories namely informal and formal technical reviews. Formal Technical reviews are sometimes called as walkthroughs or inspections. They are the most effective filter from QA standpoint. To understand the significance of these reviews, let us look at the defect amplification model shown below.

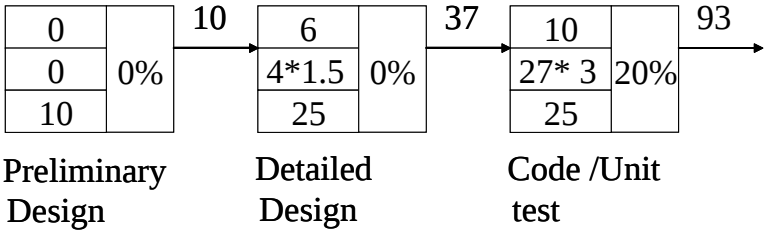
This model depicts that each development step inherits certain errors from the previous step. Some of these errors are just passed through to the next step while some are worked on and hence are amplified with a ratio of 1:x. In addition, each step may also generate some new errors. If each step has some mechanism for error detection, some of these errors may be detected and removed and the rest are passed on to the next step.

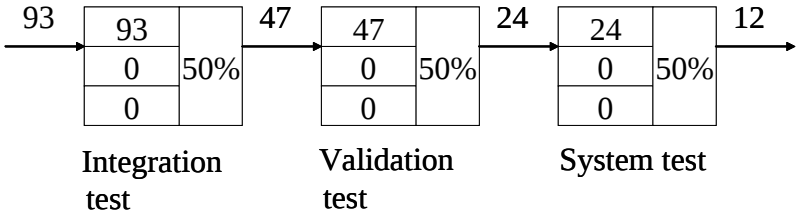


Development Step

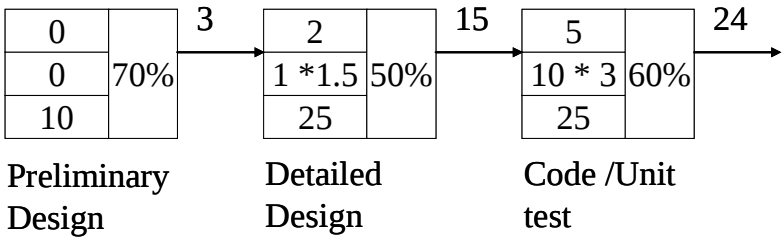
Let us now assume that we do not have any SQA related activities for the first two stages and we are only using testing for detection of any defects. Let us assume that the

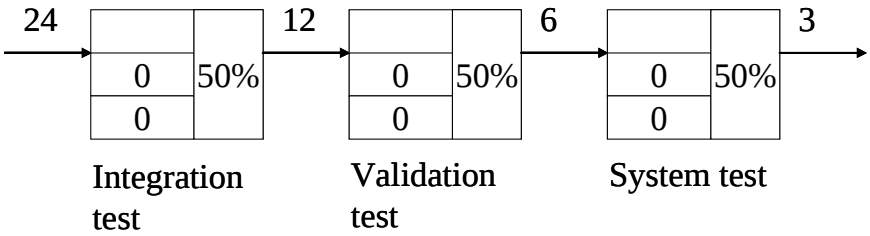
Preliminary design generated 10 defects which were passed on to Detailed design. At that phase, 6 defects were passed on to the next stages and 4 were amplified at a ration of 1:1.5. In addition, there were 25 new defects introduced at this stage. Therefore, a total of 37 defects were passed on to the next stage as shown in the diagram. In the Code and Unite test stage, we start to test our system and assuming 20% defect removal efficiency of this stage, 94 defects (80% of (10 + 27 * 3 + 25)) are passed on to the next stage. This process continues and the system is delivered with 12 defects remaining in the product.





If FTR are used in the earlier stages, the quality of the end-product is much better as shown in the following diagram. Note that in this case we have code inspection in addition to unit testing at the third stage and the defect removal efficiency of that stage is 60%.





Lecture No. 29

Formal Technical Reviews

Formal Technical Reviews are conducted by software engineers. The primary objective is to find errors during the process so that they do not become defects after release of software as they uncover errors in function, logic design, or implementation. The idea is to have early discovery of errors so they do not propagate to the next step in the process. They also ensure that the software has been represented according to predefined standards and it is developed in a uniform manner. They make projects more manageable and help groom new resources as well as provide backup and continuity.

FTRs include walkthroughs, inspections, and other small group technical assessments of software.

Guidelines for walkthroughs

FTRs are usually conducted in a meeting that is successful only if it is properly planned, controlled, and attended. The producer informs the PM that the WP is ready and the review is needed. The review meeting consists of 3-5 people and advanced preparation is required. It is important that this preparation should not require more than 2 hours of work per person. It should focus on specific (and small) part of the overall software. For example, instead of the entire design, walkthroughs are conducted for each component, or small group of components. By narrowing focus, FTR has a high probability of uncovering errors.

It is important to remember that the focus is on a work product for which the producer of the WP asks the project leader for review. Project leader informs the review leader. The review leader evaluates the WP for readiness and if satisfied generates copies of review material and distributes to reviewers for advanced preparation. The agenda is also prepared by the review leader.

Review Meetings

Review meeting is attended by the review leader, all reviewers, and the producer. One of the reviewer takes the roles of recorder. Producer walks through the product, explaining the material while other reviewers raise issues based upon their advanced preparation. When valid problems or errors are recorded, the recorder notes each one of them. At the end of the RM, all attendees of the meeting must decide whether to:

- Accept the product without further modification
- Reject the product due to severe errors
  - Major errors identified
  - Must review again after fixing
- Accept the product provisionally
  - Minor errors to be fixed
  - No further review

Review Reporting and Record keeping

© Copy Right Virtual University of Pakistan

During the FTR the recorder notes all the issues. They are summarized at the end and a review issue list is prepared. A summary report is produced that includes:

- What is reviewed
- Who reviewed it
- What were the findings and conclusions

It then becomes part of project historical record.

The review issue list

It is sometimes very useful to have a proper review issue list. It has two objectives.

- Identify problem areas within the WP
- Action item checklist

It is important to establish a follow-up procedure to ensure that items on the issue list have been properly addressed.

Review Guidelines

It is essential to note that an uncontrolled review can be worse than no review. The basis principle is that the review should focus on the product and not the producer so that it does not become personal. Remember to be sensitive to personal egos. Errors should be pointed out gently and the tone should be loose and constructive.

This can be achieved by setting an agenda and maintaining it. In order to do so, the review team should:

- Avoid drift
- Limit debate and rebuttal
- Enunciate problem areas but don't try to solve all problems
- Take written notes
- Limit the number of participants and insist upon advanced preparation
- Develop a checklist for each product that is likely to be reviewed
- Allocate resources and schedule time for FTRs
- Conduct meaningful training for all reviewers
- Review your early reviews
- Determine what approach works best for you

© Copy Right Virtual University of Pakistan

Lecture No. 30

Statistical Software Quality Assurance

Statistical SQA is a technique that measures the quality in a quantitative fashion. It implies that information about defects is collected and categorized and an attempt is made to trace each defect to underlying cause. It uses Pareto Principle to identify vital causes (80% of defects can be traced to 20% of causes) and moves to correct the problems that have caused the defects.

Example

Let us assume that information about defects is collected for one year and categorized as follows:

1. Incomplete or erroneous specifications (IES)
2. Misinterpretation of customer communication (MCC)
3. Intentional deviation from specifications (IDS)
4. Violation of programming standards (VPS)
5. Error in data representation (EDR)
6. Inconsistent component interface (ICI)
7. Error in digital logic (EDL)
8. Incomplete or erroneous testing (IET)
9. Inaccurate or incomplete documentation (IID)
10. Error in programming language translation of design (PLT)
11. Ambiguous or inconsistent HCI (HCI)
12. Miscellaneous (MIS)

The following data is collected for these categories

Error Category	**Serious**	**Moderate**	**Minor**	**Sub Total**
IES	34	68	103	205
MCC	12	68	76	156
IDS	1	24	23	48
VPS	0	15	10	25
EDR	26	68	36	130
ICI	9	18	31	58
EDL	14	12	19	45
IET	12	35	48	95
IID	2	20	14	36
PLT	15	19	26	60
HCI	3	17	8	28
MIS	0	15	41	56

© Copy Right Virtual University of Pakistan

Total	128	379	435	942

We can easily see the following:

- IES, MCC, and EDR are the vital errors – cause 53% of all errors
- IES, EDR, PLT, and EDL are vital if only serious errors are considered

We now start corrective action focused on vital few. For example for EDR we review the data representation techniques to identify the possible improvement areas and adopt a use case tool for data modeling and perform stringent data design reviews.

Error Index (EI)

Another statistical technique known as Error Index (EI) is used to develop an overall indication of improvement in software quality. The EI is computed as follows:

Let

- E_i – the total number of errors uncovered during the ith step in the SE process
- S_i – number of serious errors
- M_i – number of moderate errors
- T_i – number of minor errors
- PS – product size at the ith step
- w_s, w_m, w_t – weighting factors for serious, moderate, and minor errors. Recommended values for these are 10, 3, 1 respectively.

At each step of the software process a *Phase Index* is computed as:

$$PI_i = w_s(S_i/E_i) + w_m(M_i/E_i) + w_t(T_i/E_i)$$

Now EI is computed as the cumulative effect on each $PI_i = \sum(i \times PI_i)/PS$

It is important to note that weighting errors encountered in the SE processes more heavily than those encountered earlier. As stated earlier, it can be used to develop an overall indication of improvement in software quality.

Software Reliability

Software reliability is another very important quality factor and is defined as probability of failure free operation of a computer program in a specified environment for a specified time. For example, a program X can be estimated to have a reliability of 0.96 over 8 elapsed hours.

Software reliability can be measured, directed, and estimated using historical and development data. The key to this measurement is the meaning of term failure. Failure is defined as non-conformance to software requirements. It can be graded in many different ways as shown below:

- From annoying to catastrophic
- Time to fix from minutes to months
- Ripples from fixing

© Copy Right Virtual University of Pakistan

It is also pertinent to understand the difference between hardware and software reliability. Hardware reliability is predicted on failure due to wear rather than failure due to design. In the case of software, there is no wear and tear. The reliability of software is determined by Mean time between failure (MTBF). MTBF is calculated as:

MTBF = MTTF + MTTR

Where MTTF is the Mean Time to Failure and MTTR is the Mean time required to Repair.

Arguably MTBF is far better than defects/kloc as each error does not have the same failure rate and the user is concerned with failure and not with total error count.

A related issue is the notion of availability. It is defined as the probability that a program is operating according to requirements at a given point in time. It can be calculated as

Availability = (MTTF/MTBF) x 100

and clearly depends upon MTTR.

Lecture No. 31

Software Safety

Software Safety is a software SQA activity that focuses on identification of potential hazards that may affect software negatively and cause an entire system to fail. Modeling and analysis process is conducted as part of software safety and hazards are identified and categorized by criticality and risk.

Example

Let us assume that the following hazards are associated with a computer-based cruise control for an automobile:

- Causes uncontrolled acceleration that cannot be stopped
- Does not respond to depression of brake pedal
- Does not engage when switch is activated
- Slowly loses or gains speed

Once system-level hazards are identified, analysis techniques are used to assign severity, and probability of occurrence. This technique is similar to risk analysis. To be effective, software must be analyzed in the context of the entire system. Analysis techniques such as fault tree analysis can be used to predict the chain of events that can cause hazards and the probability that each of these events will occur to create the chain. Once hazards are identified and analyzed, safety-related requirements can be specified for the software.

Reliability and safety are closely related. Software reliability uses statistical techniques to determine the likelihood that a software failure will occur. Occurrence of a software failure does not necessarily result in a hazard or mishap. On the other hand, software safety examines the ways in which failures result in conditions that can lead to a mishap.

© Copy Right Virtual University of Pakistan

Lecture No. 32

Poka-Yoke (Mistake-Proofing)

Poka-yoke is a QA technique developed by Shingo at Toyota in 1960's. Poka-yoke devices are mechanisms that lead to prevention of potential quality problem before it occurs or the rapid detection of quality problems if they are introduced.

- Examples:
  - Light on if the car door is not properly closed
  - Warning beep if the engine is turned-off when lights are on

Characteristic of a Poka-yoke device

- It is simple and cheap
- It is part of the process
- It is located near the process task where the mistake occurs

Example of Poka-yoke in software

Let us assume that a\A software products company sells application software to an international market. The pull-down menus and associated mnemonics provided with each application must reflect the local language. For example, the English language menu item for "Close" has the mnemonic "C" associated with it. When the application is sold to Urdu speaking users, the same menu item is "Band Karen" with mnemonic "ب". To implement the appropriate menu entry for each locale, a "localizer" translates the menus accordingly. The problem is to ensure that each menu entry conforms to appropriate standards and there are no conflicts, regardless of the language used.

We can consider a prevention device to write a program to generate mnemonics automatically, given a list of labels in each menu. It would prevent mistakes, but the problem of choosing a good mnemonic is difficult and the effort required would not be justified by the benefit gained. It is therefore not a good Poka-yoke device as it is not simple.

We now consider another prevention device. In this case we write a program to prevent the localizer from choosing mnemonics that did not meet the criteria. It would prevent mistakes but the benefit is minimal as incorrect mnemonics are easy enough to detect and correct after they occur. So once again this is not a good choice.

We now consider a detection device. In this case we write a program to verify that the chosen menu labels and mnemonics meet the criteria. We run this program to detect errors. Errors are sent back to localizer. Since this device requires a lot of back and forth movement, it violates the principle of co-location.

We now modify it to the following detection device. We write a program to verify that the chosen menu labels and mnemonics meet the criteria. The localizer would run this program to detect errors. It would give quick feedback and hence it is a good Poka-yoke device.

© Copy Right Virtual University of Pakistan

Lecture No. 33

Software Configuration Management (SCM)

You may recall that software configuration management (SCM) is one of the five KPA required for an organization to be at CMM level 2. That means, according to SEI, effective project management is not possible without having a proper SCM function in place.
The basic idea behind SCM is to manage and control change. As mentioned by Bersoff, no matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle. It is therefore essential that we manage and control it in a fashion that this continuous change does not convert into chaos.
This has become more important in the context of contemporary software development as we are getting more and more complex software projects in terms of size, sophistication, and technology. In addition, these software systems are used by millions of users all over the world. These systems need multilingual and multi-platform support (hardware, software) and have to operate in a distributed environment. That means that a software system may come in many configuration flavors including desktop, standard, professional, and enterprise versions. There is a brutal competition out there and any complacency may result in losing a big market share. The huge maintenance frequency – corrective and adaptive – makes life even more difficult.
More complex development environment with shorter reaction time results in confusion and chaos!

Change Chaos
This frequent change, if not managed properly, results in chaos. First of all there would be problems of identification and tracking which would result in questions like the following:

- "This program worked yesterday. What happened?"
- "I fixed this error last week. Why is it back?"
- "Where are all my changes from last week?"
- "This seems like an obvious fix. Has it been tried before?"
- "Who is responsible for this change?"

Then there are problems of version selection. The typical problems faced are:

- "Has everything been compiled? Tested?"
- "How do I configure for test, with my updates and no others?"
- "How do I exclude this incomplete/faulty change?"
- "I can't reproduce the error in this copy!"
- "Exactly which fixes went into this configuration?"
- "Oh my God!. I need to merge 250 files!"
- Nobody knows which versions of programs are final
    - Where is the latest version?
    - Which version is the right one? I have so many
    - I have lost my latest changes
- Latest versions of code overwritten by old versions
- There was a minor problem, I fixed it but it is no longer working

© Copy Right Virtual University of Pakistan

- I can't figure-out the changes made to the previous version.
- I can't go back

Then there are software delivery problems.

- "Which configuration does this customer have?"
- "Did we deliver a consistent configuration?"
- "Did the customer modify the code?"
- "The customer skipped the previous two releases. What happens if we send him the new one?"
- Shipped the wrong version to the client.

This is not all. There may be more chaos in the following shapes and forms:

- The design document is out of sync with programs
- I don't know if all the changes that were suggested have been incorporated
- Which code was sent to testing?

SCM is a function that, if implemented, will reduce these problems to a minimal level.

Configuration management

As defined by CMM, the purpose of SCM is to establish and maintain the integrity or software products through the project's life cycle.
Configuration management is concerned with managing evolving software systems. It acknowledges that system change is a team activity and thus it aims to control the costs and effort involved in making changes to a system.
SCM involves the development and application of procedures and standards to manage an evolving software product and is part of a more general quality management process.
Baseline
When released to CM, software systems are called *baselines* and serve as the starting point for further development.A baseline is a software configuration management concept that helps us to control change without seriously impeding justifiable change. It is defined by IEEE as:

A specification or a product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
Software Configuration Item (SCI)
A Software Configuration Item is the information that is created as part of the software engineering process. Typical SCIs include requirement specifications, design specification, source code, test cases and recorded results, user guides and installation manuals, executable programs, and standards and procedures (for example C++ design guidelines).

Software Configuration Management Tasks

Software configuration management tasks include:

© Copy Right Virtual University of Pakistan

- Identification
- Version Control
- Change Control
- Configuration Auditing
- Reporting

Identification addresses how does an organization identify and manage the many existing versions of a program in a manner that will enable changes to be accommodated efficiently?

Version Control talks about how does an organization control changes before and after software is released to a customer? It is actually a combination of procedures and tools to manage different versions of the software configuration.

Clemm states that

> *Configuration management allows the user to specify alternative configurations of the software system through the selection of the appropriate versions. This is supported by associating with each software version, and then allowing configuration to be specified and constructed by describing the set of desired attributes.*

A version has many different attributes. In the simplest form a specific version number that is attached to each object and in the complex form it may have a string of Boolean variables (switches) that indicate specific types of functional changes that have been applied to the system.

The Change Control process addresses the important question of who has the responsibility for approving and ranking changes. Configuration Auditing deals with ensuring that the changes have been made properly and finally Reporting talks about the mechanism used to apprise others of changes that are made. Configuration Identification involves identification of a tool for SCM. Then a baseline is established and identified which is then used to identify configurable software items. At the minimum, all deliverables must be identified as configurable items. This includes design, software, test cases, tutorials, and user guides.

© Copy Right Virtual University of Pakistan

Lecture No. 34

Product Release Version Numbering System

Product release is the act of making a product available to its intended customers. After a product has had its first release, it enters a product release cycle. New versions of the product are made available that may fix defects or add features that were not in previous releases. These changes are categorized as updates or upgrades. An update fixes product defects. An upgrade enhances the product feature set and will include updates.

Release Numbering

Each individual product release is viewed as being in a unique state which is the total set of functionality possessed by the product release. Release numbering is a mechanism to identify the product's functionality state. Each release will have a different product state and hence will have a different release number. Although there is no industry standard, typically, a three field compound number of the format "X.Y.Z" is used. The different fields communicate functionality information about the product release.

The first digit, X, is used for the major release number which is used to identify a major increase in the product functionality. The major release number is usually incremented to indicate a significant change in the product functionality or a new product base-line.

The second digit, Y, stands for feature release number. The feature release number is iterated to identify when a set of product features have been added or significantly modified from their originally documented behaviour.

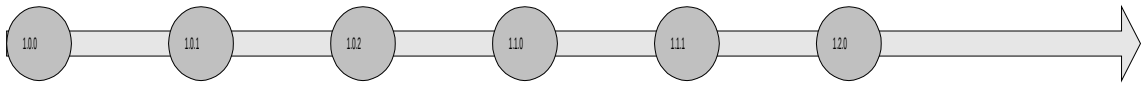
The third digit, Z, is called the defect repair number and is incremented when a set of defects is repaired. Defect repair/maintenance is considered to be any activity that supports the release functionality specification and it may a fix for some bugs or some maintenance to enhance the performance of the application.

Conventionally, a release number starts with a major number of one, followed by zero for its feature and maintenance numbers. This results in a release number 1.0.0. If the first new release that is needed is a defect repair release, the last digit would be iterated to one, resulting in 1.0.1. If two additional defect repair releases are needed, we would eventually have a release number of 1.0.3. Assume that an upgrade feature release is now needed. We will need to iterate the second field and will roll back the defect repair number to 0, resulting in a release number of 1.1.0. When we iterate the major release identifier, both the feature and defect numbers would be reset back to zero.

Internal Release Numbering

A special type of release is internal release. Internal releases are used by the development organization as a staging mechanism for functionality. The most common internal releases are the regular builds. A common way to number internal builds is to use the same release number that would be used for final release with some additional information added to it to identify the built. It is suggested that we add an extra (fourth) field to identify and keep track of internal builds.

The following diagram depicts the above mentioned version numbering system.



© Copy Right Virtual University of Pakistan

Change control

James Back points out the difficulties related to change control as follows:

Change control is vital. But the forces that make it necessary also make it annoying. We worry about change because a tiny perturbation in the code can cause a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities. We worry about change because a single rogue developer could sink the project; yet brilliant ideas originate in the minds of those rogues, and a burdensome change control process could discourage them from doing creative work.

That is, like all engineering activities, change control is the name of a balancing act. Too much or too little change control creates different types of problems as uncontrolled change rapidly leads to chaos.

The change control process is elaborated in the following sub-section.

© Copy Right Virtual University of Pakistan

Lecture No. 35

Change Control Process

The first component of the change control process is the Change Control Authority (CCA) or a Change Control Board (CCB). A CCA or CCB includes people from both developer and client side.

Whenever a change is required, the CCB decides whether to allow this change to happen or deny it. If it is decided that a change is needed, an Engineering Change Order or ECO is generated. An ECO defines the change to be made, the constraints that must be respected, and the criteria for review and audit. The change control process thus involves the following steps.

1) need for change is recognized
2) change request from user
3) developer evaluates
4) change report is generated
5) change control authority (CCA) decides
6) Either step 6a or 6b is performed. Steps numbers 7 to 17 are performed only if step 6b is performed.
   a)
      i) change request is denied
      ii) user is informed
      iii) no further action is taken.
   b) assign people to SCIs
7) check-out SCIs
8) make the change
9) review/audit the change
10) check-in SCIs
11) establish a “baseline” for testing
12) perform SQA and testing activities
13) check-in the changed SCIs
14) promote SCI for inclusion in next release
15) rebuild appropriate version
16) review/audit the change
17) include all changes in release

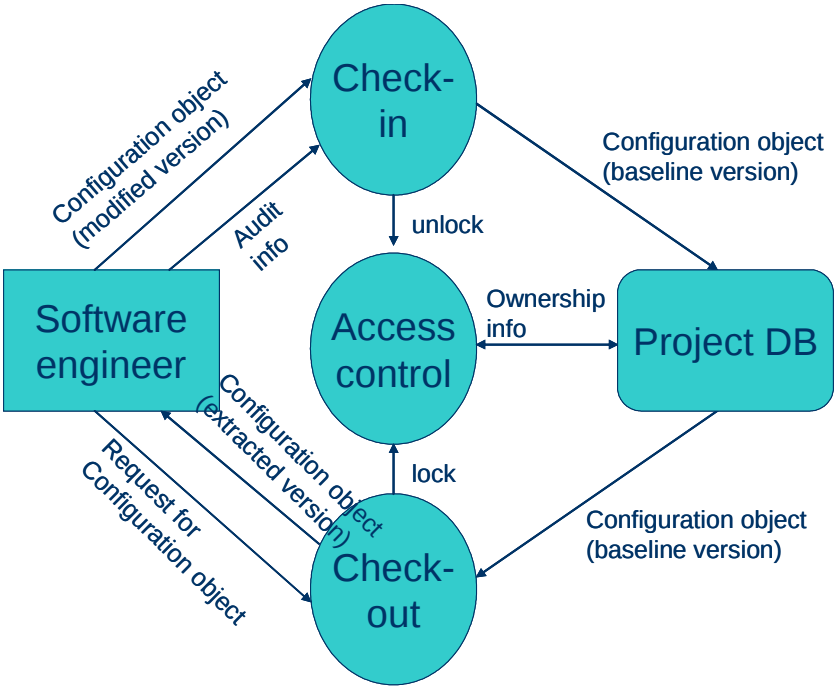
Thus, a change is incorporated in a controlled and strict manner.

Check-in and check-out

In SCM, the processes of Check-in and Check-out take a central stage. These are two important elements of change control and provide access and synchronization control. Access control manages who has the authority to check-out the object and synchronization control ensures that parallel changes by two different people do not overwrite one another.

© Copy Right Virtual University of Pakistan

Synchronization control implements a control on updates. When a copy is checked-out, other copies can be checked out for use only but they cannot be modified. In essence, it implements a single-writer multiple-readers protocol. This process is depicted in the following diagram.



Configuration Audit

Configuration audit ensures that a change has been properly implemented. It involves formal technical reviews and software configuration audit. Configuration audit assess a configuration object for characteristics that are generally not considered during audit. It is conducted by the SQA group. It looks into the following aspects of the change:

- Has the change specified in the ECO been made? Have any additional modifications been incorporated?
- Has a FTR been conducted to assess technical correctness?
- Has the software process been followed?
- Have the SE standards been properly applied?
- Has the change been highlighted in the SCI?
    - Change date and author
    - Have the attributes of the configuration object been updated?
- Have the SCM procedures for noting the change, recording it, and reporting it been followed?
- Have all related SCI's been properly updated?

An audit report is finally generated and any non-compliances are highlighted so that they may be corrected.

© Copy Right Virtual University of Pakistan

Configuration Status reporting (CSR)

Configuration Status Reporting (CSR) is also known as status accounting. It reports on the following items:

- What happened?
- Who did it?
- When did it happen?
- What else will be affected?

If it is not done then the organization faces the left hand not knowing what the right hand is doing syndrome. Without it, if a person responsible for the change leaves for whatever reason, it would be difficult to understand the whole scenario.

CSR reports are generated on regular basis. Each time an SCI is assigned a new identification, a CSR entry is made. Each time a change is approved by CCA, a CSR entry is made. Also, each time configuration audit is conducted, the results are reported as part of CSR task.

© Copy Right Virtual University of Pakistan

Lecture No. 36

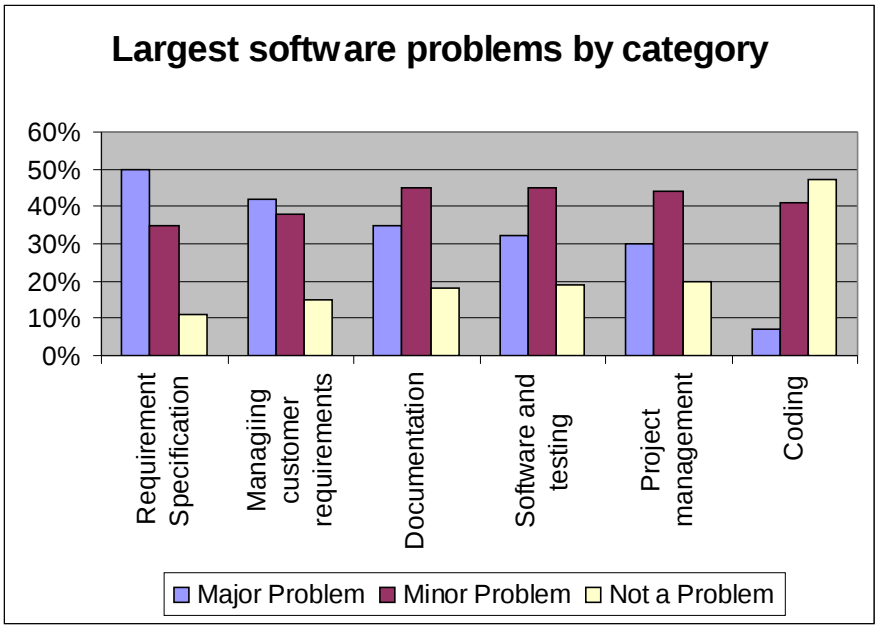
Requirement Management and CMM

CM standards

CM should always be based on a set of standards which are applied within an organisation. Standards should define how items are identified, how changes are controlled and how new versions are managed. Standards may be based on external CM standards (e.g. IEEE standard for CM ANSI/IEEE Std. No. 828-1983, 1042-1987, 1028-1988). Existing standards are based on a waterfall process model - new standards are needed for evolutionary development.

The Requirement Problem

The goal of software development is to develop quality software – on time and on budget – that meets customers' real needs. Project success depends on good requirement management. It may be recalled that requirement errors are the most common type of software development errors and the most costly to fix. It may also be recalled that requirement errors are listed as one of the roots causes of software project failure. According to a Standish Group report, lack of user input is responsible for13% of all project failures; incomplete requirements and specifications for 12% of all project failures; and changing requirements are responsible for 12% of all project failures.



The above graph shows the result of an industry survey. It is clear to see that requirement specifications are considered to be the most significant cause of major software problems by a majority of practitioners.

© Copy Right Virtual University of Pakistan

Requirement Management

Requirement management is also one of the 5 KPA defined at CMM level 2. Without having a proper requirement management function, chances of a project's success are slim.

Requirements Management KPA Goals statement says that:

1. The software requirements are controlled to establish a baseline for software engineering and management use.
2. Software plans, products, and activities are kept consistent with the software requirements.

Requirement Management is defined as a systematic approach to eliciting, organizing, and documenting the requirements of the system, and a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.

It includes establishing and maintaining an agreement with the customer on the requirement for the software project. It involves

- Defining the requirement baseline
- Reviewing proposed requirement changes and evaluating the likely impact of each proposed change before deciding whether to approve it
- Incorporating approved requirement changes in the project in a controlled manner
- Keeping project plans current with the requirements
- Negotiating new commitments based on estimated impact on changed requirements
- Tracing individual requirements to their corresponding design, source code, and test cases
- Tracking requirement status and change activity throughout the project

Requirement Attributes long question in 2023

We need to tag requirements with certain attributes in order to manage them in an orderly fashion. Attributes are used to establish a context and background for each requirement. They go beyond the description of intended functionality. They can be used to filter, sort, or query to view selected subset of the requirements. A list of possible attributes is enumerated as below:

1. Requirement ID
2. Creation date
3. Created by
4. Last modified on
5. Last modified by
6. Version number
7. Status
8. Origin
9. Subsystem

© Copy Right Virtual University of Pakistan

10. Product Release
11. Priority

Requirement Status

The requirement status attribute is one of the most useful ones. It can be used to keep track of different requirements going through different phases. The possible status values are proposed, approved, implemented, verified, and deleted. These are elaborated in the following paragraphs.

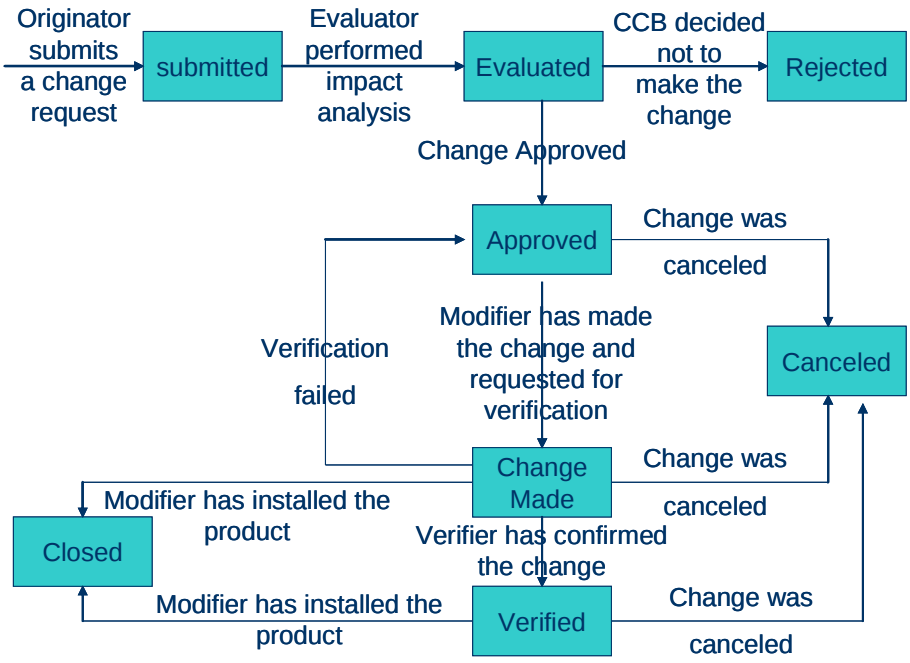
1. Proposed: The requirement has been requested by a source who has the authority to provide requirements.

2. Approved: The requirement has been analyzed, its impact on the rest of the project has been estimated, and it has been allocated to the baseline for a specific build number or product release. The software development group has committed to implement the requirement.
3. Implemented: The code that implements the requirement has been designed, written, and unit tested.

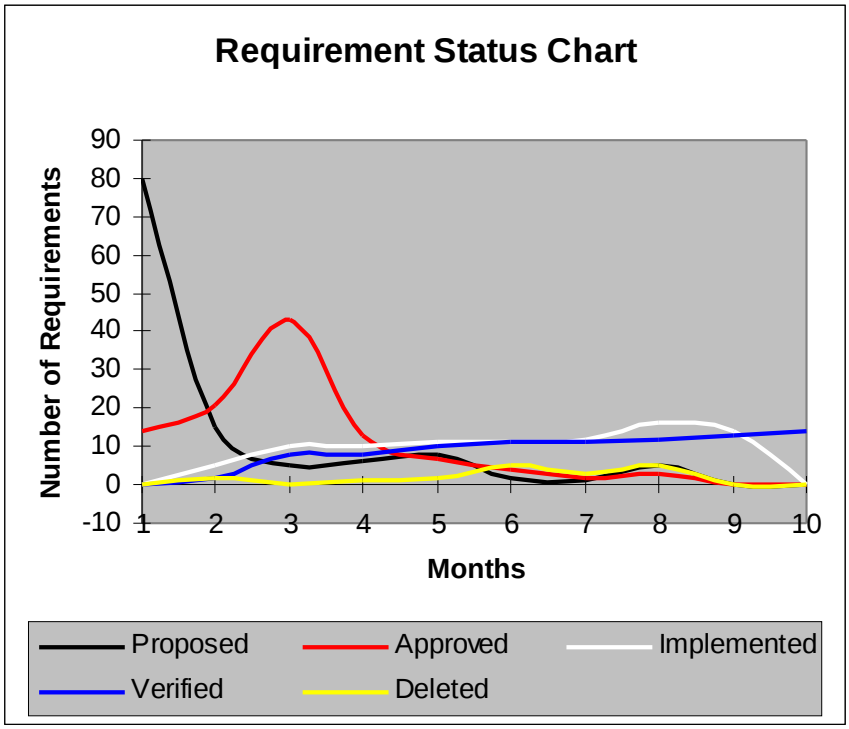
4. Verified: The implemented requirement has been verified through the selected approach, such as testing or inspection. The requirement has been traced to pertinent test cases. The requirement is now considered complete.
5. Deleted: A planned requirement has been deleted from the baseline. Include an explanation of why and by whom the decision was made to delete the requirement.

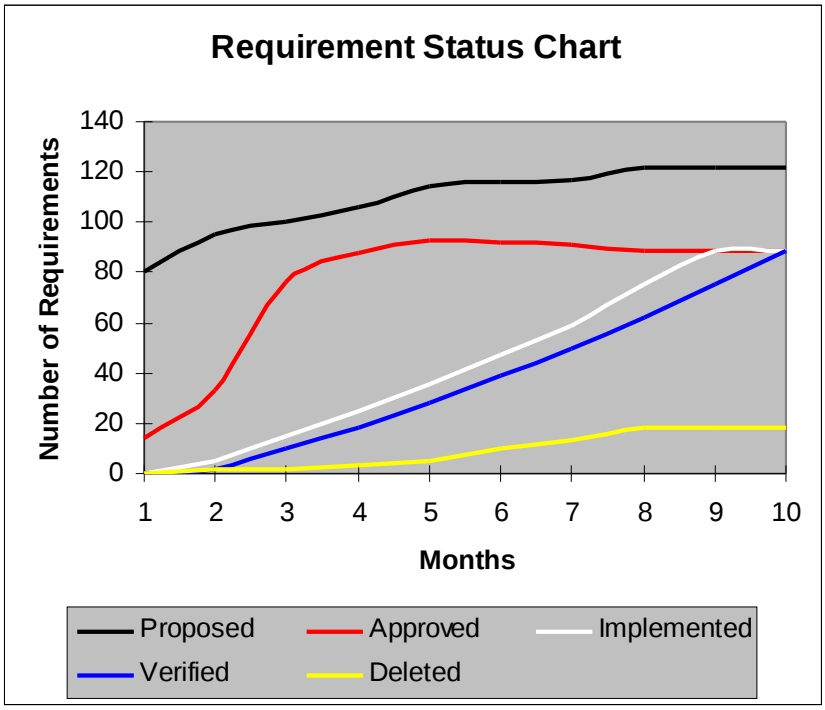
Change Request Status

As the requirement go through different phases, their status is updated accordingly. The following state transition diagram captures the sequence of requirement status changes.



The changes in the requirement status can be plotted as shown below to get an idea of the stability of the requirements and the progress of the project. It is easy to see that it is normal to have unstable requirements in the beginning but if they requirements stayed volatile till the end then the progress would be slow.





Managing Scope Creep

We must always remember that requirements will change, no matter what**.** That means we have to be able to manage changing requirements. Software organizations and professionals must learn to manage changing requirements. A major issue in requirements engineering is the rate at which requirements change once the requirements phase has "officially" ended. We therefore need to try to take it to a minimum level. For that we need to measure the change activity.

© Copy Right Virtual University of Pakistan

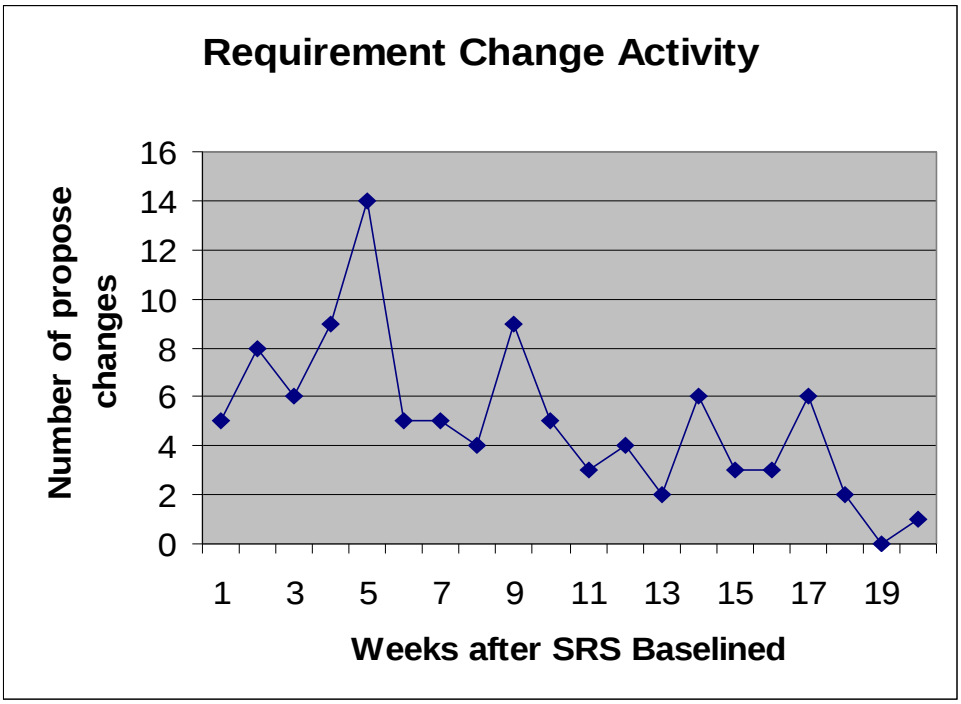
Lecture No. 37

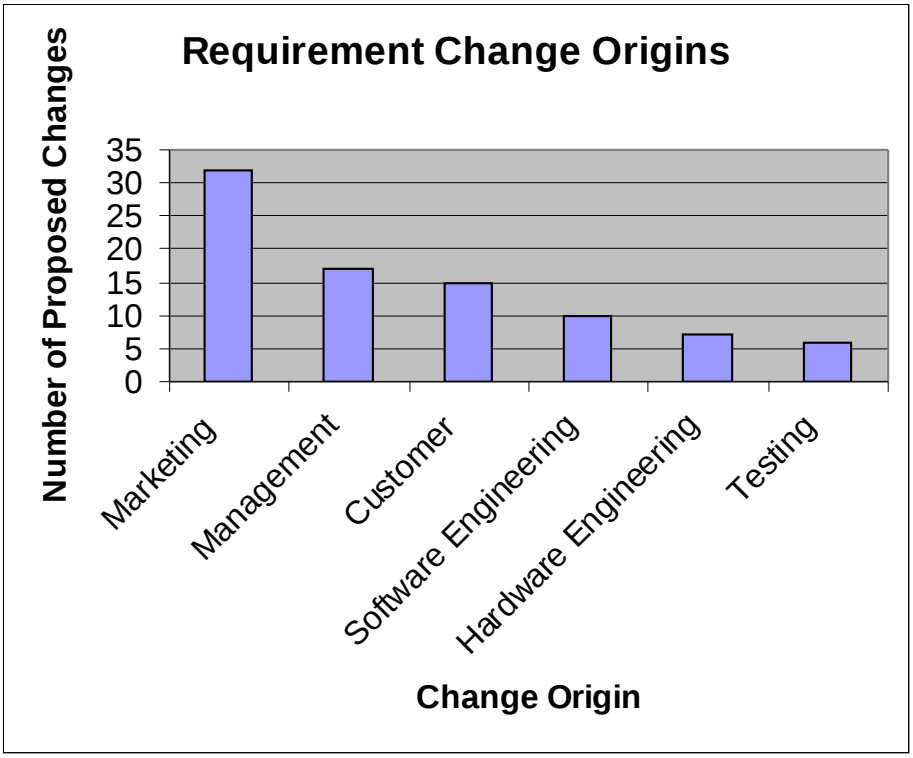
Measuring Change Activity

Measurement of change activity is a way to assess the stability of the requirements and to identify opportunities for process improvement. In this regards, the following could be measured

- The number of change requests received, open, and closed
- The cumulative number of changes made including added, deleted, and modified requirements
- The number of change requests that originated from each source
- The number of changes proposed and made in each requirement since it was base-lined
- The total effort devoted to handling changes

These can then be plotted as shown in the graphs below to get an idea of the stability of the systems. It is important to note that the sooner the requirements come to a stable state after establishing the baseline the better it is for the project. It is also useful to track the source of the requirement changes so that processes governing those areas causing more frequent changes may be strengthened in future projects.





Requirement Traceability

Requirement traceability is a very important consideration for requirement management. It is really hard to manage requirements that are not traceable.

A Software Requirement Specification (SRS) is traced if the origin of its requirements is clear. That means that the SRS includes references to earlier supportive documents. An SRS is traceable if it written in a manner that facilitates the referencing of each individual requirement stated therein.

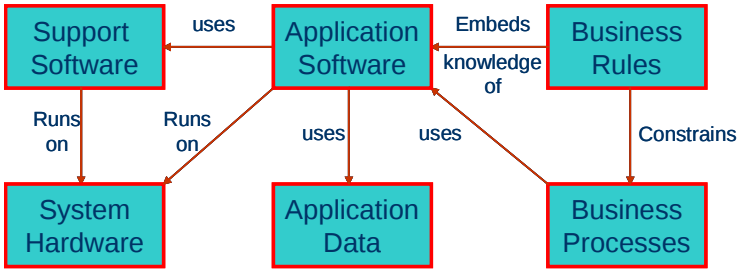
It is important to trace requirements both ways. That is from origin of a requirement to how it is implemented. This is a continuous process. It is also important that the rationale of requirements must also be traced. Traceability is important for the purposes of certification, change impact analysis, maintenance, project tracking, reengineering, reuse, risk reduction, and testing. That is it plays an important role in almost every aspect of the project and its life cycle.

© Copy Right Virtual University of Pakistan

Lecture No. 38 important

Legacy systems

A system is considered to be a legacy system if it has been in operation for many years. A legacy system has many components. These include business processes, business rules, application software, application data, support software, and system hardware. The relationship among these components is shown in the following diagram.



Maintaining Legacy System

Maintaining legacy system is expensive. It is often the case that different parts of the system have been implemented by different teams, lacking consistency. Part or all of the system may be implemented using an obsolete language. In most cases system documentation is inadequate and out of date. In some cases the only documentation is the source code. In some cases even the source code is not available.

Many years of maintenance have usually corrupted the system structure, making it increasingly difficult to understand. The data processed by the system may be maintained in different files which have incompatible structures. There may be data duplication and the documentation of the data itself may be out of date, inaccurate, and incomplete.

As far as the system hardware is concerned, the hardware platform may be outdated and is hard to maintain. In many cases, the legacy systems have been written for mainframe hardware which is no longer available, expensive to maintain, and not be compatible with current organizational IT purchasing policies.

Support software includes OS, database, and compiler etc. Like hardware, it may be obsolete and no longer supported by the vendors.

A time therefore comes when an organization has to make this decision whether to keep the old legacy system or to move it to new platform and environment. Moving it to new environment is known as legacy system migration.

© Copy Right Virtual University of Pakistan

Legacy migration risks

Legacy system migration however is not an easy task and there are a number of risks involved that need to be mitigated. First of all, there is rarely a complete specification of the system available. Therefore, there is no straight forward way of specifying the services provided by a legacy system. Thus, important business rules are often embedded in the software and may not be documented elsewhere. Business processes and the way legacy systems operate are often intertwined. New software development may take several years.

New software development is itself risky as changes to one part of the system inevitably involve further changes to other components.

We therefore need to assess a legacy system before a decision for migration is made.

Legacy System Assessment long question in current papers 2023

For each legacy system, there are four strategic options:

1. Scrap the system completely: This is the case when system is not making an effective contribution to business processes and business processes have changed significantly and the organization is no longer completely dependent upon the system.

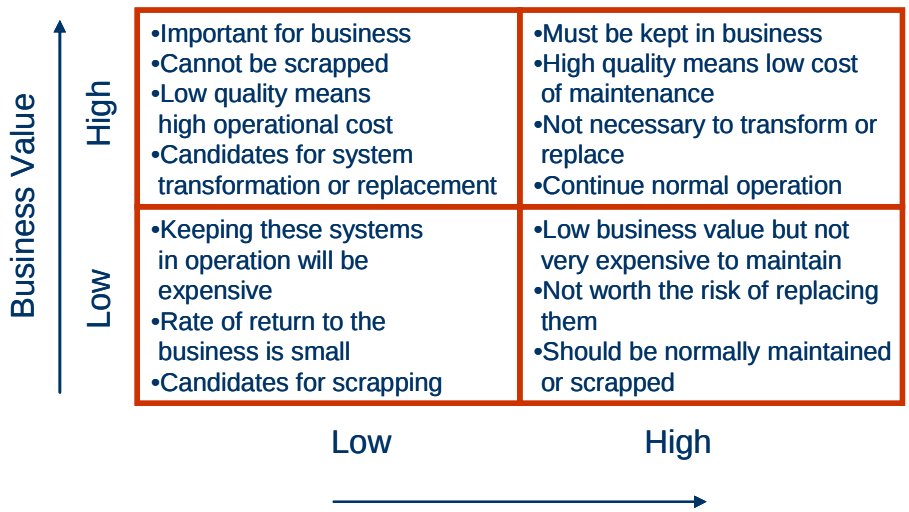
2. Continue maintaining the system: This option is used when system is still required, it is stable, and requirements are not changing frequently

3. Transform the system in some way to improve its maintainability: this option is exercised when system quality has been degraded and regular changes to the system are required.

Replacement:

4. Replace the system with a new system: this path is taken when old system cannot continue in operation and off-the shelf alternative is available or system can be developed at a reasonable cost.

For these decisions, a legacy system can be assessed from two different perspectives – business value and quality. The following four quadrant assessment matrix can be used for this purpose.



© Copy Right Virtual University of Pakistan

Business Value Assessment

It is important to note that this is a subjective judgment and requires different business viewpoints. These view points include end-users, customers, line managers, IT managers, and senior managers.

End Users assess the system from the perspective of how effective do they find the system in supporting their business processes and how much of the system functionality is used.

The customers look at the system and ask is the use of the system transparent to customer or are their interaction constrained by the system, are they kept waiting because of the system, and do system errors have a direct impact on the customer.

From an IT Manager's perspective the following questions need to be asked: Are there difficulties in finding people to work on the system? Does the system consume resources which could be deployed more effectively on other systems?

Line Managers ask: do managers think that the system is effective in contributing to success of their unit? Is the cost of keeping the system in use justified? Is the data managed by the system critical for the functioning of the manager's unit?

Senior Managers look at the system from the angle that does the system and associated business process make an effective contribution to the business goal?

© Copy Right Virtual University of Pakistan

Lecture No. 39

Environment Assessment

The legacy system also needs to be assessed from an environment's perspective. This involves looking at the supplier, failure rate, age, performance, support requirements, maintenance cost, and interoperability.

These angles are elaborated in the following paragraph:

Supplier stability: Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, is the system maintained by someone else?

Failure rate: Does the hardware have a high rate of reported failure? Does the support software crash often and force system restarts?

Age: How old is the hardware and software?

Performance: Is the performance of the system adequate? Do performance problems have a significant effect on system users?

Support requirements: What local support is required by hardware and software? If there are high costs associated with this support, it may be worth considering system replacement?

Maintenance Cost: What are the costs of hardware maintenance and software licenses?
Interoperability: Are there problems interfacing the system with other systems? Can compilers etc be used with current versions of the operating system? Is system emulation required?

Application software assessment

The application software is assessed on the following parameters:

Understandability: How difficult is it to understand the software code of the current system? How complex are the control structures that are used?

Documentation: What system documentation is available? Is the documentation complete, consistent, and up-to-date?

Data: Is there an explicit data model for the system? To what extent is data duplicated in different files?

© Copy Right Virtual University of Pakistan

Programming Language: Are modern compilers available for the programming language? Is the language still used for new system development?

Test Data: Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?

Personnel skills: Are there people available who have the skills to maintain the system?

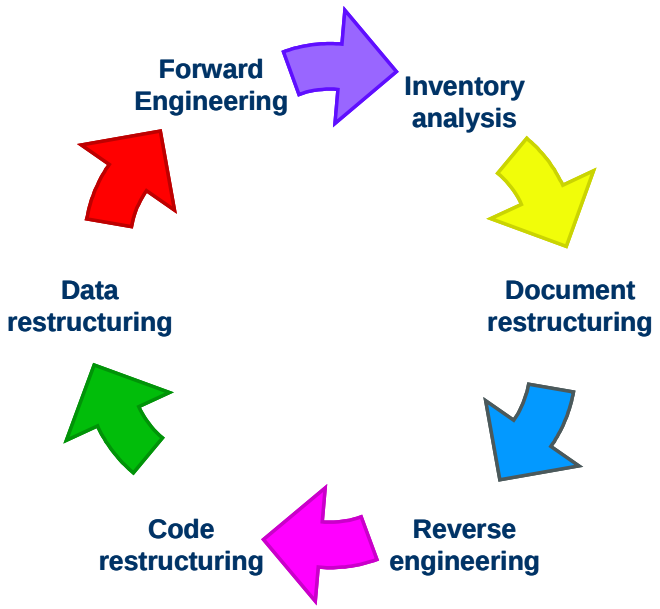
As migration is a very costly and risky business, the decision to migrate the system is made after assessing it from all these angles and it is determined that the time has come to migrate this system and it is worth spending the required amount of money and time for undertaking that effort.

Software Reengineering

Software solutions often automate the business by implementing business rules and business processes. In many cases, the software makes the business processes. As these rules and processes change, the software must also change. A time comes when these changes become very difficult to handle. So reengineering is re-implementing legacy systems to make them more maintainable. It is a long term activity.

Software Reengineering Process Model

The software reengineering is a non-trivial activity. Just like legacy migration, careful analysis must be carried out before a decision for reengineering is taken. The following process model can be used to reengineer a legacy system.



Inventory analysis

Inventory analysis is the first step in the reengineering process. At this stage, inventory of all applications is taken a note of their size, age, business criticality, and current maintainability is made. Inventory should be updated regularly as the status of the application can change as a function of time.

© Copy Right Virtual University of Pakistan

Document restructuring

The next step in the reengineering process is document restructuring. Weak documentation is a trademark of many legacy applications. Without proper documentation, the hidden rules, business processes, and data cannot be easily understood and reengineered.

In this regards, the following options are available:

1. Create documentation: Creating documentation from scratch is very time consuming. If program is relatively stable and is coming to the end of its useful life then just leave it as it is.
2. Update documentation: This option also needs a lot of resources. A good approach would be to update documentation when the code is modified.

Reverse engineering

Reverse engineering is the next step in the process. Reverse engineering for software is a process for analyzing a program in an effort to create a representation of the program at a higher level of abstraction than the source code. Reverse engineering is the process of design recovery. At this stage, documentation of the overall functionality of the system that is not there is created. The overall functionality of the entire system must be understood before more detailed analysis can be carried out.

Reverse engineering activities include:

- Reverse engineering to understand processing
- Reverse engineering to understand data
  - Internal data structures
  - Database structures
- Reverse engineering user interfaces

Program Restructuring

Program is restructured after the reverse engineering phase. In this case we modify source code and data in order to make it amenable to future changes. This includes code as well as data restructuring. Code restructuring requires redesign with same function with higher quality than original program and data restructuring involves restructuring the database or the database schema. It may also involve code restructuring.

© Copy Right Virtual University of Pakistan

Lecture No. 40

Forward Engineering

At the end forward engineering is carried out. It means incorporating the new business processes and rules in the system. Forward engineering requires application of SE principles, methods, and concepts to re-create an existing application. In most cases forward engineering does not simply create a modern equivalent of an older program, rather new user and technology requirements are integrated into the reengineering effort.

The Economics of Reengineering

As reengineering is a costly and risky undertaking, a cost benefit analysis for the reengineering effort must be carried out.

This analysis is carried out in the following manner.

Let

P1 **:** current annual maintenance cost for an application
P2 **:** current annual operation cost for an application
P3 **:** current annual business value of an application
P4 **:** predicted annual maintenance cost after reengineering
P5 **:** predicted annual operations cost after reengineering
P6 **:** predicted annual business value cost after reengineering
P7 **:** estimated reengineering cost
P8 **:** estimated reengineering calendar time
P9 **:** reengineering risk factor (1.0 is nominal)
L **:** expected life of the system

Now the cost of maintenance is calculated as:

$$C_{maintenance} = [P3 – (P1 + P2)] \times L$$

Cost of reengineering would then be given by the formula:

$$C_{reengineering} = [P6 – (P4 + P5) \times (L – P8) – (P7 \times P9)]$$

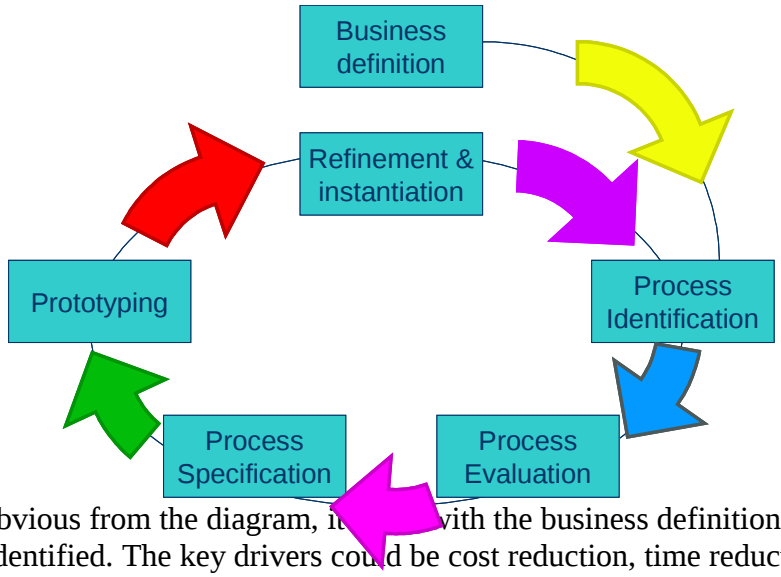
© Copy Right Virtual University of Pakistan

Lecture No. 41

Business Process Reengineering

A concept similar to software reengineering is of business process reengineering (BPR). A business process is "a set of logically related tasks performed to achieve a defined business outcome". It is the way certain business is conducted. Purchasing services and supplies, hiring new employees, paying suppliers are examples of business processes.

For BPR the following process model may be used.



As obvious from the diagram, it starts with the business definition where business goals are identified. The key drivers could be cost reduction, time reduction, quality improvement, and personnel development and empowerment. It may be defined at the business level or for a specific component of the business.

The next step is process identification. At this time processes that are critical to achieving the goals are identified and are ranked by importance, and need for change.

The short listed processes are then evaluated. Existing process is analyzed and measured and process tasks are identified. The cost and time consumed is measured as well as the quality and performance problems are identified.

Then, process specification and design is carried out. Use cases are prepared for each process to be redesigned and a new set of tasks are designed for the processes and then they are prototyped.

A redesigned business process must be prototyped before it is fully integrated into the business.

Based on the feedback the business process is refined.

© Copy Right Virtual University of Pakistan

Lecture No. 42

Software Refactoring

Software refactoring is the process of changing a software system such that the external behavior of the system does not change while the internal structure of the system is improved. This is sometimes called "Improving the design after it has been written".

Fowler defines refactoring as A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. It is achieved by applying a series of refactorings without changing its observable behavior.

A (Very) Simple Example

Let us consider a very simple example. The refactoring involved in this case is known as "Consolidate Duplicate Conditional Fragments". As the name suggests, this refactoring lets the programmer improve the quality of the code by grouping together the duplicate code, resulting in less maintenance afterwards.

```
if (isSpecialDeal()) {
   total = price * 0.95;
   send ();
}
else {
   total = price * 0.98;
   send ();
}
```

In this case, *send* is being called from different places. We can consolidate as follows:

```
if (isSpecialDeal())
   total = price * 0.95;
else
   total = price * 0.98;

send ();
```

It can further be improved if we calculate total outside the if statement as shown below.

```
if (isSpecialDeal())
   factor = 0.95;
else
   factor = 0.98;

total = price * factor;
send ();
```

Although this is a trivial example, it nevertheless is useful and teaches us how can be consolidate code by grouping together pieces of code from different segments.

© Copy Right Virtual University of Pakistan

Refactoring: Where to Start?

The first question that we have to ask ourselves is: how do you identify code that needs to be refactored? Fowler et al has devised a heuristic based approach to this end known as "Bad Smells" in Code. The philosophy is simple: ***"if it stinks, change it".***

Bad Smells in Code

They have identified many different types of "bad smells". These are briefly described in the following paragraphs:

- Duplicated Code
  - bad because if you modify one instance of duplicated code but not the others, you (may) have introduced a bug!
- Long Method
  - long methods are more difficult to understand; performance concerns with respect to lots of short methods are largely obsolete
- Large Class
  - Large classes try to do too much, which reduces cohesion
- Long Parameter List
  - hard to understand, can become inconsistent
- Divergent Change
  - Deals with cohesion; symptom: one type of change requires changing one subset of methods; another type of change requires changing another subset
- Shotgun Surgery
  - a change requires lots of little changes in a lot of different classes
- Feature Envy
  - A method requires lots of information from some other class (move it closer!)
- Data Clumps
  - attributes that clump together but are not part of the same class
- Primitive Obsession
  - characterized by a reluctance to use classes instead of primitive data types
- Switch Statements
  - Switch statements are often duplicated in code; they can typically be replaced by use of polymorphism (let OO do your selection for you!)
- Parallel Inheritance Hierarchies
  - Similar to Shotgun Surgery; each time I add a subclass to one hierarchy, I need to do it for all related hierarchies
- Lazy Class
  - A class that no longer "pays its way". e.g. may be a class that was downsized by refactoring, or represented planned functionality that did not pan out
- Speculative Generality
  - "Oh I think we need the ability to do this kind of thing someday"
- Temporary Field

© Copy Right Virtual University of Pakistan

  - An attribute of an object is only set in certain circumstances; but an object should need all of its attributes
- Message Chains
  - a client asks an object for another object and then asks that object for another object etc. Bad because client depends on the structure of the navigation
- Middle Man
  - If a class is delegating more than half of its responsibilities to another class, do you really need it?
- Inappropriate Intimacy
  - Pairs of classes that know too much about each other's private details
- Alternative Classes with Different Interfaces
  - Symptom: Two or more methods do the same thing but have different signature for what they do
- Incomplete Library Class
  - A framework class doesn't do everything you need
- Data Class
  - These are classes that have fields, getting and setting methods for the fields, and nothing else; they are data holders, but objects should be about data AND behavior
- Refused Bequest
  - A subclass ignores most of the functionality provided by its superclass
- Comments (!)
  - Comments are sometimes used to hide bad code
  - "…comments often are used as a deodorant" (!)

Breaking a Method

We have already seen example of duplicate code. We now look at another simple example of long method. Although, in this case, the code is not really long, it however demonstrates how longer segments of code can be broken into smaller and more manageable (may be more reusable as well) code segments.

The following code segment sorts an array of integers using "selection sort" algorithm.

```
for (i=0; i < N-1; i++) {
        min = i;
        for (j = i; j < N; j++)
                if (a[j] < a[min]) min = j;
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
}
```

We break it into smaller fragments by making smaller functions out of different steps in the algorithm as follows:

```
int minimum (int a[ ], int from, int to)
{
        int min = from;
        for (int i = from; i <= to; i++)
                if (a[i] < a[min]) min = i;
        return min;
}

void swap (int &x, int &y)
{
        int temp = x;
        x = y;
        y = temp;
}
```

The sort function now becomes simpler as shown below.

```
for (i=0; i < N-1; i++) {
        min = minimum (a, i, N-1);
        swap(a[i], a[min]);
}
```

It can be seen that it is now much easier to understand the code and hence is easier to maintain. At the same time we have got two separate reusable functions that can be used elsewhere in the code.

A slightly more involved example
(It has mostly been adapted from Fowler's introduction to refactoring which is freely available on the web)

Let us consider a simple program for a video store. It has three classes: Movie, Rental, and Customer. Program is told which movies a customer rented and for how long and it then calculates the charges and the Frequent renter points. Customer object can print a statement (in ASCII).

Here is the code for the classes. DomainObject is a general class that does a few standard things, such as hold a name.

```
public class DomainObject {

        public DomainObject (String name)   {
                _name = name;
        };

        public DomainObject ()      {};

        public String name ()       {
                return _name;
        };
```

```
        public String toString() {
                return _name;
        };

        protected String _name = "no name";
}
```

Movie represents the notion of a film. A video store might have several tapes in stock of the same movie

```
public class Movie extends DomainObject {
   public static final int  CHILDRENS = 2;
   public static final int  REGULAR = 0;
   public static final int  NEW_RELEASE = 1;


        private int _priceCode;

        public Movie(String name, int priceCode) {
                _name = name;
                _priceCode = priceCode;
        }

        public int priceCode() {
                return _priceCode;
        }

        public void persist() {
                Registrar.add ("Movies", this);
        }

        public static Movie get(String name) {
                return (Movie) Registrar.get ("Movies", name);
        }
}
```

The movie uses a class called a registrar (not shown) as a class to hold instances of movie. I also do this with other classes. I use the message persist to tell an object to save itself to the registrar. I can then retrieve the object, based on its name, with a get(String) method.

The tape class represents a physical tape.

```
class Tape extends DomainObject
   {
   public Movie movie() {
        return _movie;
   }
   public Tape(String serialNumber, Movie movie) {
```

© Copy Right Virtual University of Pakistan

```
        _serialNumber = serialNumber;
        _movie = movie;
}
private String _serialNumber;
private Movie _movie;
}
```

The rental class represents a customer renting a movie.

```
class Rental extends DomainObject
{
public int daysRented() {
        return _daysRented;
}
public Tape tape() {
        return _tape;
}
private Tape _tape;
public Rental(Tape tape, int daysRented) {
        _tape = tape;
        _daysRented = daysRented;
}
private int _daysRented;
}
```

The customer class represents the customer. So far all the classes have been dumb encapsulated data. Customer holds all the behavior for producing a statement in its statement() method.

```
class Customer extends DomainObject
{
public Customer(String name) {
   _name = name;
}
public String statement() {
   double totalAmount = 0;
   int frequentRenterPoints = 0;
   Enumeration rentals = _rentals.elements();
   String result = "Rental Record for " + name() + "\n";
   while (rentals.hasMoreElements()) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.nextElement();

      //determine amounts for each line
      switch (each.tape().movie().priceCode()) {
         case Movie.REGULAR:
            thisAmount += 2;
            if (each.daysRented() > 2)
               thisAmount += (each.daysRented() - 2) * 1.5;
            break;
```

```
            case Movie.NEW_RELEASE:
               thisAmount += each.daysRented() * 3;
               break;
            case Movie.CHILDRENS:
               thisAmount += 1.5;
               if (each.daysRented() > 3)
                  thisAmount += (each.daysRented() - 3) * 1.5;
               break;

         }
         totalAmount += thisAmount;

         // add frequent renter points
         frequentRenterPoints ++;
         // add bonus for a two day new release rental
         if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

         //show figures for this rental
         result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(thisAmount) +
"\n";

      }
      //add footer lines
      result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
      return result;

   }
   public void addRental(Rental arg) {
         _rentals.addElement(arg);
   }
   public static Customer get(String name) {
         return (Customer) Registrar.get("Customers", name);
   }
   public void persist() {
         Registrar.add("Customers", this);
   }
   private Vector _rentals = new Vector();
   }
```

What are your impressions about the design of this program? I would describe as not well designed, and certainly not object-oriented. For a simple program is this, that does not really matter. Thereís nothing wrong with a quick and dirty *simple* program. But if we imagine this as a fragment of a more complex system, then I have some real problems with this program. That long statement routine in the Customer does far too much. Many of the things that it does should really be done by the other classes.

© Copy Right Virtual University of Pakistan

This is really brought out by a new requirement, just in from the users, they want a similar statement in html. As you look at the code you can see that it is impossible to reuse any of the behavior of the current statement() method for an htmlStatement(). Your only recourse is to write a whole new method that duplicates much of the behavior of statement(). Now of course this is not too onerous. You can just copy the statement() method and make whatever changes you need. So the lack of design does not do too much to hamper the writing of htmlStatement(), (although it might be tricky to figure out exactly where to do the changes). But what happens when the charging rules change? You have to fix both statement() and htmlStatement(), and ensure the fixes are consistent. The problem from cut and pasting code comes when you have to change it later. Thus if you are writing a program that you donít expect to change, then cut and paste is fine. If the program is long lived and likely to change, then cut and paste is a menace.

But you still have to write the htmlStatement() program. You may feel that you should not touch the existing statement() method, after all it works fine. Remember the old engineering adage: "if it ainít broke, donít fix it". statement() may not be broke, but it does hurt. It is making your life more difficult to write the htmlStatement() method.

So this is where refactoring comes in. When you find you have to add a feature to a program, and the programís code is not structured in a convenient way to add the feature; then first refactor the program to make it easy to add the feature, then add the feature.

Extracting the Amount Calculation

The obvious first target of my attention is the overly long statement() method. When I look at a long method like that, I am looking to take a chunk of the code an *extract a method* from it.

Extracting a method is taking the chunk of code and making a method out of it. An obvious piece here is the switch statement (which I'm highlighting below).

```
public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + name() + "\n";
      while (rentals.hasMoreElements()) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();

         //determine amounts for each line
         switch (each.tape().movie().priceCode()) {
            case Movie.REGULAR:
               thisAmount += 2;
               if (each.daysRented() > 2)
                  thisAmount += (each.daysRented() - 2) * 1.5;
               break;
            case Movie.NEW_RELEASE:
               thisAmount += each.daysRented() * 3;
               break;
```

```
            case Movie.CHILDRENS:
               thisAmount += 1.5;
               if (each.daysRented() > 3)
                  thisAmount += (each.daysRented() - 3) * 1.5;
               break;

         }
         totalAmount += thisAmount;

         // add frequent renter points
         frequentRenterPoints ++;
         // add bonus for a two day new release rental
         if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

         //show figures for this rental
         result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(thisAmount) +
"\n";

      }
      //add footer lines
      result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
      return result;

   }
```

This looks like it would make a good chunk to extract into its own method. When we extract a method, we need to look in the fragment for any variables that are local in scope to the method we are looking at, that local variables and parameters. This segment of code uses two: each and thisAmount. Of these each is not modified by the code but thisAmount is modified. Any non-modified variable we can pass in as a parameter. Modified variables need more care. If there is only one we can return it. The temp is initialized to 0 each time round the loop, and not altered until the switch gets its hands on it. So we can just assign the result. The extraction looks like this.

```
public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + name() + "\n";
      while (rentals.hasMoreElements()) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();

         //determine amounts for each line
         thisAmount = amountOf(each);
         totalAmount += thisAmount;
```

```
        // add frequent renter points
        frequentRenterPoints ++;
        // add bonus for a two day new release rental
        if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

        //show figures for this rental
        result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(thisAmount) +
"\n";

      }
      //add footer lines
      result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
      return result;

   }

   private int amountOf(Rental each) {
      int thisAmount = 0;
      switch (each.tape().movie().priceCode()) {
         case Movie.REGULAR:
            thisAmount += 2;
            if (each.daysRented() > 2)
               thisAmount += (each.daysRented() - 2) * 1.5;
            break;
         case Movie.NEW_RELEASE:
            thisAmount += each.daysRented() * 3;
            break;
         case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (each.daysRented() > 3)
               thisAmount += (each.daysRented() - 3) * 1.5;
            break;

      }
      return thisAmount;
   }
```

When I did this the tests blew up. A couple of the test figures gave me the wrong answer. I was puzzled for a few seconds then realized what I had done. Foolishly I had made the return type of amountOf() int instead of double.

```
   private double amountOf(Rental each) {
      double thisAmount = 0;
      switch (each.tape().movie().priceCode()) {
         case Movie.REGULAR:
            thisAmount += 2;
            if (each.daysRented() > 2)
```

```
            thisAmount += (each.daysRented() - 2) * 1.5;
         break;
      case Movie.NEW_RELEASE:
         thisAmount += each.daysRented() * 3;
         break;
      case Movie.CHILDRENS:
         thisAmount += 1.5;
         if (each.daysRented() > 3)
            thisAmount += (each.daysRented() - 3) * 1.5;
         break;

   }
   return thisAmount;
}
```

Itís the kind of silly mistake that I often make, and it can be a pain to track down as Java converts ints to doubles without complaining (but merrily rounding). Fortunately it was easy to find in this case, because the change was so small. Here is the essence of the refactoring process illustrated by accident. Because each change is so small, any errors are very easy to find. You don't spend long debugging, even if you are as careless as I am.

This refactoring has taken a large method and broken it down into two much more manageable chunks. We can now consider the chunks a bit better. I don't like some of the variables names in amountOf() and this is a good place to change them.

```
private double amountOf(Rental aRental) {
   double result = 0;
   switch (aRental.tape().movie().priceCode()) {
      case Movie.REGULAR:
         result += 2;
         if (aRental.daysRented() > 2)
            result += (aRental.daysRented() - 2) * 1.5;
         break;
      case Movie.NEW_RELEASE:
         result += aRental.daysRented() * 3;
         break;
      case Movie.CHILDRENS:
         result += 1.5;
         if (aRental.daysRented() > 3)
            result += (aRental.daysRented() - 3) * 1.5;
         break;

   }
   return result;
}
```

Is that renaming worth the effort? Absolutely. Good code should communicate what it is doing clearly, and variable names are key to clear code. Never be afraid to change the names to things to improve clarity. With good find and replace tools, it is usually not difficult. Strong typing and testing will highlight anything you miss. Remember any fool

can write code that a computer can understand, good programmers write code that humans can understand.

Lecture No. 43

Moving the amount calculation

As I look at amountOf, I can see that it uses information from the rental, but does not use information from the customer. This method is thus on the wrong object, it should be moved to the rental. To *move a method* you first copy the code over to rental, adjust it to fit in its new home and compile.

```
Class Rental
   double charge() {
      double result = 0;
      switch (tape().movie().priceCode()) {
         case Movie.REGULAR:
            result += 2;
            if (daysRented() > 2)
               result += (daysRented() - 2) * 1.5;
            break;
         case Movie.NEW_RELEASE:
            result += daysRented() * 3;
            break;
         case Movie.CHILDRENS:
            result += 1.5;
            if (daysRented() > 3)
               result += (daysRented() - 3) * 1.5;
            break;

      }
      return result;
   }
```

In this case fitting into its new home means removing the parameter.

The next step is to find every reference to the old method, and adjusting the reference to use the new method. In this case this step is easy as we just created the method and it is in only one place. In general, however, you need to do a find across all the classes that might be using that method.

```
public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + name() + "\n";
      while (rentals.hasMoreElements()) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();
```

© Copy Right Virtual University of Pakistan

```
        //determine amounts for each line
        thisAmount = each.charge();
        totalAmount += thisAmount;

        // add frequent renter points
        frequentRenterPoints ++;
        // add bonus for a two day new release rental
        if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

        //show figures for this rental
        result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(thisAmount) +
"\n";

    }
    //add footer lines
    result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
    result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
    return result;

  }
```

When I've made the change the next thing is to remove the old method. The compiler should then tell me if I missed anything.

There is certainly some more I would like to do to Rental.charge() but I will leave it for the moment and return to Customer.statement().

The next thing that strikes me is that thisAmount() is now pretty redundant. It is set to the result of each.charge() and not changed afterwards. Thus I can eliminate thisAmount by *replacing a temp with a query*.

```
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Enumeration rentals = _rentals.elements();
    String result = "Rental Record for " + name() + "\n";
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();

        //determine amounts for each line
        totalAmount += each.charge();

        // add frequent renter points
        frequentRenterPoints ++;
        // add bonus for a two day new release rental
        if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;
```

```
            //show figures for this rental
            result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(each.charge())
+ "\n";

        }
        //add footer lines
        result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
        result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
        return result;

    }
```

I like to get rid of temporary variables like thus as much as possible. Temps are often a problem in that they cause a lot of parameters to get passed around when they don't need to. You can easily lose track of what they are there for. They are particularly insidious in long methods. Of course there is a small performance price to pay, here the charge is now calculated twice. But it is easy to optimize that in the rental class, and you can optimize much more effectively when the code is properly refactored.

Extracting Frequent Renter Points

The next step is to do a similar thing for the frequent renter points. Again the rules vary with the tape, although there is less variation than with the charging. But it seems reasonable to put the responsibility on the rental. First we need to extract a method from the frequent renter points part of the code (highlighted below).

```
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + name() + "\n";
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();

            //determine amounts for each line
            totalAmount += each.charge();

            // add frequent renter points
            frequentRenterPoints ++;
            // add bonus for a two day new release rental
            if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

            //show figures for this rental
            result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(each.charge())
+ "\n";

        }
```

```
        //add footer lines
        result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
        result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
        return result;

    }
```

Again we look at the use of locally scoped variables. Again it uses each, which can be passed in as a parameter. The other temp used is frequentRenterPoints. In this case frequentRenterPoints does have a value beforehand. The body of the extracted method doesn't read the value, however, so we don't need to pass it in as a parameter as long as we use an appending assignment.

```
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + name() + "\n";
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();

            //determine amounts for each line
            totalAmount += each.charge();

            // add frequent renter points
            frequentRenterPoints += frequentRenterPointOf(each);

            //show figures for this rental
            result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(each.charge())
+ "\n";

        }
        //add footer lines
        result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
        result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
        return result;

    }

int frequentRenterPointOf(Rental each) {
        if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) return 2;
        else return 1;
    }
```

I did the extraction, compiled and tested, and then did a move. With refactoring small steps are the best, that way less tends to go wrong.

© Copy Right Virtual University of Pakistan

```
public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + name() + "\n";
      while (rentals.hasMoreElements()) {
         Rental each = (Rental) rentals.nextElement();

         //determine amounts for each line
         totalAmount += each.charge();

         // add frequent renter points
         frequentRenterPoints += each.frequentRenterPoints();

         //show figures for this rental
         result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(each.charge())
+ "\n";

      }
      //add footer lines
      result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
      return result;

   }

int frequentRenterPoints() {
      if ((tape().movie().priceCode() == Movie.NEW_RELEASE) && daysRented() > 1)
return 2;
      else return 1;
   }
```

Removing Temps

As I suggested before, temporary variables can be a problem. They are only useful within their own routine, and thus they encourage long complex routines. In this case we have two temporary variables, both of which are being used to get a total from the rentals attached to the customer. Both the ascii and html versions will require these totals. I like to *replace temps with queries*. Queries are accessible to any method in the class, and thus encourage a cleaner design without long complex methods.

I began by replacing totalAmount with a charge() method on customer.

```
   public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + name() + "\n";
      while (rentals.hasMoreElements()) {
```

```
        Rental each = (Rental) rentals.nextElement();

        // add frequent renter points
        frequentRenterPoints += each.frequentRenterPoints();

        //show figures for this rental
        result += "\t" + each.tape().movie().name()+ "\t" + String.valueOf(each.charge())
+ "\n";

      }
      //add footer lines
      result +=  "Amount owed is " + String.valueOf(charge()) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter
points";
      return result;

   }

   private double charge(){
      double result = 0;
      Enumeration rentals = _rentals.elements();
      while (rentals.hasMoreElements()) {
         Rental each = (Rental) rentals.nextElement();
         result += each.charge();
      }
      return result;
   }
```

After compiling and testing that refactoring, I then did the same for frequentRenterPoints.

```
   public String statement() {
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + name() + "\n";
      while (rentals.hasMoreElements()) {
         Rental each = (Rental) rentals.nextElement();
         //show figures for each rental
         result += "\t" + each.tape().movie().name()+ "\t" +
                   String.valueOf(each.charge()) + "\n";
      }
      //add footer lines
      result +=  "Amount owed is " + String.valueOf(charge()) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints()) +
                " frequent renter points";
      return result;
   }

   private int frequentRenterPoints() {
      int result = 0;
      Enumeration rentals = _rentals.elements();
      while (rentals.hasMoreElements()) {
```

© Copy Right Virtual University of Pakistan

```
        Rental each = (Rental) rentals.nextElement();
        result += each.frequentRenterPoints();
    }
    return result;
}
```

It is worth stopping and thinking a bit about this refactoring. Most refactoring reduce the amount of code, but this one increases it. That's because Java requires a lot of statements to set up a summing loop. Even a simple summing loop with one line of code per element needs six lines of support around it. Itís an idiom that is obvious to any programmer but it is noise that hides what the intent of the loop is. As Java develops and builds up its ability to handle block closures in the style of Smalltalk, I expect that overhead to decrease, probably to the single line that such an expression would take in Smalltalk.

The other concern with this refactoring lies in performance. The old code executed the while loop once, the new code executes it three times. If the while loop takes time, this might significantly impair performance. Many programmers would not do this refactoring simply for this reason. But note the words "if" and "might". While some loops do cause performance issues, most do not. So while refactoring donít worry about this. When you optimize you will have to worry about it, but you will then be in a much better position to do something about it, and you will have more options to optimize effectively. (For a good discussion on why it is better to write clearly first and then optimize, see [McConnell, Code Complete].

These queries are now available to any code written in the customer class. Indeed they can easily be added to the interface of the class should other parts of the system need this information. Without queries like these, other methods need to deal with knowing about the rentals and building the loops. In a complex system that will lead to much more code to write and maintain.

You can see the difference immediately with the htmlStatement(). I am now at the point where I take off my refactoring hat and put on my adding function hat. I can write htmlStatement()like this (and add an appropriate test).

```
public String htmlStatement() {
    Enumeration rentals = _rentals.elements();
    String result = "<H1>Rentals for <EM>" + name() + "</EM></H1><P>\n";
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        //show figures for each rental
        result += each.tape().movie().name()+ ": " +
                String.valueOf(each.charge()) + "<BR>\n";
    }
    //add footer lines
    result +=  "<P>You owe <EM>" + String.valueOf(charge()) + "</EM><P>\n";
    result += "On this rental you earned <EM>" +
String.valueOf(frequentRenterPoints()) + "</EM> frequent renter points<P>";
    return result;

}
```

© Copy Right Virtual University of Pakistan

There is still some code copied from the ascii version, but that is mainly due to setting up the loop. Further refactoring could clean that up further, extracting methods for header, footer, and detail line are one route I could take. But that isnít where I want to spend my time, I would like to move onto the methods Iíve moved onto rental. Back on with the refactoring hat.

Moving the Rental Calculations to Movie

Yes itís that switch statement that is bugging me. It is a bad idea to do a switch based on an attribute of another object. If you must use a switch statement, it should be on your own data, not on someone elseís. This implies that the charge should move onto movie

```
Class movie Ö
   double charge(int daysRented) {
      double result = 0;
      switch (priceCode()) {
         case REGULAR:
            result += 2;
            if (daysRented > 2)
               result += (daysRented - 2) * 1.5;
            break;
         case NEW_RELEASE:
            result += daysRented * 3;
            break;
         case CHILDRENS:
            result += 1.5;
            if (daysRented > 3)
               result += (daysRented - 3) * 1.5;
            break;

      }
      return result;
   }
```

For this to work I have to pass in the length of the rental, which of course is data due of the rental. The method effectively uses two pieces of data, the length of the rental and the type of the movie. Why do I prefer to pass the length of rental rather than the movieís type? Its because type information tends to be more volatile. I can easily imagine new types of videos appearing. If I change the movieís type I want the least ripple effect, so I prefer to calculate the charge within the movie.

I compiled the method into movie and then adjusted the charge method on rental to use the new method.

```
Class rentalÖ
   double charge() {
      return _tape.movie().charge(_daysRented);
   }
```

© Copy Right Virtual University of Pakistan

Some people would prefer to remove that chain of calls by having a charge(int) message on tape. This would lead to

```
Class rental
   double charge() {
      return _tape.charge(_daysRented);
   }

Class tape
   double charge() {
      return _movie.charge(_daysRented);
   }
```

You can make that change if you like, I donít tend to worry about message chains providing that they all lie in the same package. If they cross package boundaries, then Iím not so happy, and would add an insulating method.

Having done this with charge amounts, Iím inclined to do the same with frequent renter points. The need is less pressing, but I think it is more consistent to do them both the same way. And again if the movie classifications change it makes it easier to update the code.

```
Class rentalÖ
   int frequentRenterPoints() {
      return _tape.movie().frequentRenterPoints(_daysRented);
   }

class movieÖ
   int frequentRenterPoints(int daysRented){
      if ((priceCode() == NEW_RELEASE) && daysRented > 1) return 2;
      else return 1;
   }
```

With these two changes I can hide those constants, which is generally a Good Thing. Even constant data should be private.

```
   private static final int  CHILDRENS = 2;
   private static final int  REGULAR = 0;
   private static final int  NEW_RELEASE = 1;
```

To really do this, however, I need to change a couple of other parts of the class. I need to change how we create a movie. I used to create a movie with a message like

```
new Movie ("Ran", Movie.REGULAR);
```

and the constructor

```
class MovieÖ
        private Movie(String name, int priceCode) {
                _name = name;
```

```
                _priceCode = priceCode;
        }
```

To keep this type code hidden I need some creation methods.

```
public static Movie newNewRelease(String name){
   return new Movie (name, NEW_RELEASE);
}
public static Movie newRegular(String name){
   return new Movie (name, REGULAR);
}
public static Movie newChildrens(String name) {
   return new Movie (name, CHILDRENS);
}
```

Now I create a new movie with

```
Movie.newRegular("Monty Python and the Holy Grail");
```

Movies can change their classification. I change a movieís classification with

```
aMovie.setPriceCode(Movie.REGULAR);
```

I will need to add a bunch of methods to handle the changes of classification.

```
public void beRegular() {
   _priceCode = REGULAR;
}

public void beNewRelease() {
   _priceCode = NEW_RELEASE;
}

public void beChildrens() {
   _priceCode = CHILDRENS;
}
```

Itís a bit of effort to set up these methods, but they are a much more explicit interface then the type codes. Just looking at the name of the method tells you what kind of movie you are getting. This makes the code more understandable. The trade off is that each time I add a price code I have to add a creation and update method. If I had lots of price codes this would hurt (so I wouldnít do it). If I have a few, however, then itís quite reasonable.

At lastÖ inheritance

So we have several types of movie, which have different ways of answering the same question. This sounds like a job for subclasses. We could have three subclasses of movie, each of which can have its own version of charge.

© Copy Right Virtual University of Pakistan

This would allow me to replace the switch statement by using polymorphism. Sadly it has one slight flaw: it doesnít work. A move can change its classification during its lifetime. An object cannot change its class during its lifetime. There is a solution however, the *state pattern* [Gang of Four]. With the state pattern the classes look like this.

By adding the indirection we can do the subclassing from the price code object, changing the price whenever we need to.

With a complex class you have to move data and methods around in small pieces to avoid errors, it seems slow but it is the quickest because you avoid debugging. For this case I could probably move the data and methods in one go as the whole thing is not too complicated. However Iíll do it the bit by bit way, so you can see how it goes. Just remember to do it one small bit at a time if you do this to a complicated class.

The first step is to create the new classes. Then I need to sort out how they are managed. As the diagram shows they are all singletons. It seems sensible to get hold of them via the superclass with a method like Price.regular(). I can do this by getting the superclass to manage the instances of the subclasses.

```
abstract class Price {
   static Price regular() {
      return _regular;
   }

   static Price childrens() {
      return _childrens;
   }
   static Price newRelease() {
      return _newRelease;
   }

   private static Price _childrens = new ChildrensPrice();
   private static Price _newRelease = new NewReleasePrice();
   private static Price _regular = new RegularPrice();
}
```

Now I can begin to move the data over. The first piece of data to move over is the price code. Of course Iím not actually going to use the price code within the Price object, but I will give it the illusion of doing so. That way the old methods will still work. They key is to modify those methods that access and update the price code value within Movie. My first step is to *self-encapsulate the type code*, ensuring that all uses of the type code go though getting and setting methods. Since most of the code came from other classes, most methods already use the getting method. However the constructors do access the price code, I can use the setting methods instead.

```
        public static Movie newNewRelease(String name){
```

```
    Movie result = new Movie (name);
    result.beNewRelease();
    return result;
        }
        public static Movie newRegular(String name){
    Movie result = new Movie (name);
    result.beRegular();
    return result;
        }
        public static Movie newChildrens(String name) {
    Movie result = new Movie (name);
    result.beChildrens();
    return result;
        }

        private Movie(String name) {
                _name = name;
        }
```

After compiling and testing I now change getting and setting methods to use the new class.

```
  public void beRegular() {
     _price = Price.regular();
  }

  public void beNewRelease() {
     _price = Price.newRelease();
  }

  public void beChildrens() {
    _price = Price.childrens();
  }
  public int priceCode() {
     return _price.priceCode();
  }
```

And provide the priceCode methods on Price and its subclasses.

```
Class PriceÖ
  abstract int priceCode();

Class RegularPriceÖ
   int priceCode(){
     return Movie.REGULAR;
   }
```

To do this I need to make the constants non-private again. This is fine, I donít mind them having a little fame before they bite the dust.

© Copy Right Virtual University of Pakistan

I can now compile and test and the more complex methods donít realize the world has changed.

After moving the data I can now start moving methods. My prime target is the charge() method. It is simple to move.

```
Class MovieÖ
   double charge(int daysRented) {
      return _price.charge(daysRented);
   }

Class PriceÖ
   double charge(int daysRented) {
      double result = 0;
      switch (priceCode()) {
         case Movie.REGULAR:
            result += 2;
            if (daysRented > 2)
               result += (daysRented - 2) * 1.5;
            break;
         case Movie.NEW_RELEASE:
            result += daysRented * 3;
            break;
         case Movie.CHILDRENS:
            result += 1.5;
            if (daysRented > 3)
               result += (daysRented - 3) * 1.5;
            break;

      }
      return result;
   }
```

Once it is moved I can start *replacing the case statement with inheritance*. I do this by taking one leg of the case statement at a time, and creating an overriding method. I start with RegularPrice.

```
Class RegularPriceÖ
   double charge(int daysRented){
      double result = 2;
      if (daysRented > 2)
         result += (daysRented - 2) * 1.5;
      return result;
   }
```

This will override the parent case statement, which I just leave as it is. I compile and test for this case, then take the next leg, compile and testÖ. (To make sure Iím executing the subclass code, I like to throw in a deliberate bug and run it to ensure the tests blow up. Not that Iím paranoid or anything.)

© Copy Right Virtual University of Pakistan

```
Class ChildrensPrice
   double charge(int daysRented){
      double result = 1.5;
      if (daysRented > 3)
         result += (daysRented - 3) * 1.5;
      return result;
   }

Class NewReleasePriceÖ
   double charge(int daysRented){
      return daysRented * 3;
   }
```

When Iíve done that with all the legs, I declare the Price.charge() method abstract.

```
Class PriceÖ
   abstract double charge(int daysRented);
```

I can now do the same procedure with frequentRenterPoints(). First I move the method over to Price.

```
Class MovieÖ
   int frequentRenterPoints(int daysRented){
      return _price.frequentRenterPoints(daysRented);
   }
Class PriceÖ
   int frequentRenterPoints(int daysRented){
      if ((priceCode() == Movie.NEW_RELEASE) && daysRented > 1) return 2;
      else return 1;
   }
```

In this case, however I wonít make the superclass method abstract. Instead I will create an overriding method for new releases, and leave a defined method (as the default) on the superclass.

```
Class NewReleasePrice
   int frequentRenterPoints(int daysRented){
      return (daysRented > 1) ?
         2:
         1;
   }

Class PriceÖ
   int frequentRenterPoints(int daysRented){
      return 1;
   }
```

Now I have removed all the methods that needed a price code. So I can get rid of the price code methods and data on both Movie and Price.

© Copy Right Virtual University of Pakistan

Putting in the state pattern was quite an effort, was it worth it? The gain is now that should I change any of priceís behavior, add new prices, or add extra price dependent behavior; it will be much easier to change. The rest of the application does not know about the use of the state pattern. For the tiny amount of behavior I currently have it is not a big deal. But in a more complex system with a dozen or so price dependent methods this would make a big difference. All these changes were small steps, it seems slow to write it like this, but not once did I have to open the debugger. So the process actually flowed quite quickly.

© Copy Right Virtual University of Pakistan

Lecture No. 44

Capability Maturity Model Integration (CMMI)

Capability Maturity Model or CMM is a reference model of mature practices in a specified discipline, used to assess a group's capability to perform that discipline. In fact there are a number of CMMs. They differ by discipline (software, systems, acquisition, etc.), structure (staged versus continuous), how maturity is defined (process improvement path), and how capability is defined (institutionalization). Hence "Capability Maturity Model®" and CMM® are used by the Software Engineering Institute (SEI) to denote a particular class of maturity models.

Software CMM, the one we saw earlier on, is one of the many CMM models.

Name	Structure	Domain
Software CMM	staged	software development
System Engineering CMM	continuous	system engineering
Software Acquisition CMM	staged	software acquisition
System Security Engineering CMM	continuous	security engineering
FAA-iCMM	continuous	software engineering, systems engineering, and acquisition
IPD-CMM	hybrid	integrated product development
People CMM	staged	workforce
SPICE Model	continuous	software development

Since these models have different structure and application domains, an organization could potentially use many of these models for their different activities, at times it could become problematic for them.

CMMI Integrates systems and software disciplines into single process improvement framework and provides a framework for introducing new disciplines as needs arise. It can now be applied to just the software engineering projects in an organization or for the entire spectrum of activities outlined above.

CMMI Representations

A representation allows an organization to pursue different improvement objectives. There are two types of representations in the CMMI models: staged and continuous. The organization and presentation of the data are different in each representation. However, the content is the same.

Staged Representation

Staged representation is classical representation we have already seen previously. It:

- Provides a proven sequence of improvements, each serving as a foundation for the next.
- Permits comparisons across and among organizations by the use of maturity levels.

© Copy Right Virtual University of Pakistan

- Provides an easy migration from the SW-CMM to CMMI.
- Provides a single rating that summarizes appraisal results and allows comparisons among organizations

This representation indicates maturity of an organization's standard process -- to answer, "What is a good order for approaching improvement across the organization?"

You may recall that a maturity level is a well-defined evolutionary plateau of process improvement. There are five maturity levels and each level is a layer in the foundation for continuous process improvement using a proven sequence of improvements, beginning with basic management practices and progressing through a predefined and proven path of successive levels.

Each maturity level provides a necessary foundation for effective implementation of processes at the next level. Higher level processes have less chance of success without the discipline provided by lower levels. The effect of innovation can be obscured in a noisy process.

Higher maturity level processes may be performed by organizations at lower maturity levels, with the risk of not being consistently applied in a crisis.

Continuous Representation

Continuous representation allows you to select the order of improvement that best meets your organization's business objectives and mitigates your organization's areas of risk. It enables comparisons across and among organizations on a process-area-by-process-area basis and provides an easy migration from EIA 731 (and other models with a continuous representation) to CMMI.

As opposed to the staged representation, it indicates improvement within a single process area -- to answer, "What is a good order for approaching improvement of this process area?"

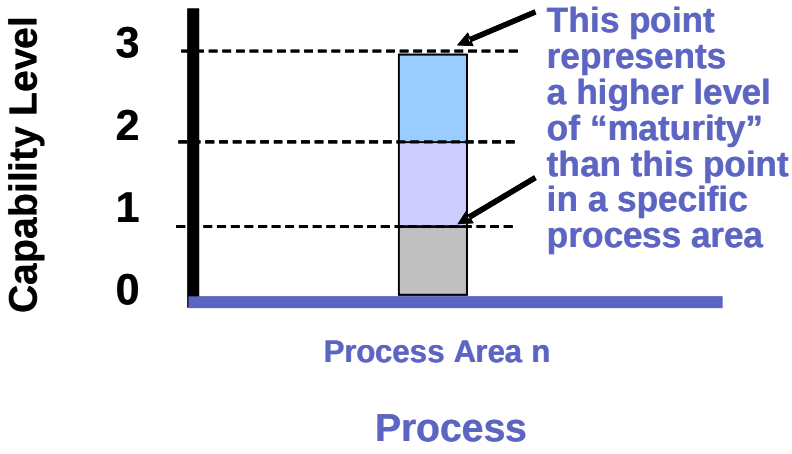
Capability Levels

A capability level is a well-defined evolutionary plateau describing the organization's capability relative to a process area. There are six capability levels. For capability levels 1-5, there is an associated generic goal. Each level is a layer in the foundation for continuous process improvement. Thus, capability levels are cumulative, i.e., a higher capability level includes the attributes of the lower levels. The five (actually six) capability levels (starting from 0) are enumerated below in the reverse order, 5 being the highest and 0 being the lowest.

5 Optimizing
4 Quantitatively Managed
3 Defined
2 Managed
1 Performed
0 Incomplete

© Copy Right Virtual University of Pakistan

The process area capability of an implemented process can be represented by a bar as shown below.



Relating Process Area Capability and Organizational Maturity

Organizational maturity is the focus of the staged representation, whereas process area capability is the focus of the continuous representation.

Organizational maturity and process area capability are similar concepts. The difference between them is that organizational maturity pertains to a set of process areas across an organization, while process area capability deals with a set of processes relating to a single process area or specific practice.

Comparison of Representations

Staged

- Process improvement is measured using maturity levels.
- Maturity level is the degree of process improvement across a predefined set of process areas.
- Organizational maturity pertains to the "maturity" of a set of processes across an organization

Continuous

- Process improvement is measured using capability levels.
- Capability level is the achievement of process improvement within an individual process area.
- Process area capability pertains to the "maturity" of a particular process across an organization.

© Copy Right Virtual University of Pakistan

Advantages of Each Representation

Staged provides a roadmap for implementing groups of process areas and sequencing of implementation. It has a familiar structure for those transitioning from the Software CMM.

Continuous provides maximum flexibility for focusing on specific process areas according to business goals and objectives and has a familiar structure for those transitioning from EIA 731.

As the staged representation requires all KPAs to be addressed at a particular level before a company can move to the next maturity level, it may not be easy for small companies to implement this model. There may be a number of activities that may not be relevant to their type of work but they would still have to do them in order to be at a certain level. On the other hand, organization can focus on their own areas of expertise and may be able to achieve high capability levels in some areas without bothering about the rest. This is a great advantage for small organization and hence this model is believed to be more suitable for small Pakistani organizations than the staged one.

© Copy Right Virtual University of Pakistan

Lecture No. 45

CMM Overview

CMM Maturity Levels

There are five levels defined along the continuum of the CMM and, according to the SEI: "Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief."

Level 1 - Ad hoc (Chaotic)

It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an *ad hoc*, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes.

Organizational implications

(a) Institutional knowledge tends to be scattered (there being limited structured approach to knowledge management) in such environments, not all of the stakeholders or participants in the processes may know or understand all of the components that make up the processes. As a result, process performance in such organizations is likely to be variable (inconsistent) and depend heavily on the institutional knowledge, or the competence, or the heroic efforts of relatively few people or small groups.

(b) Despite the chaos, such organizations manage to produce products and services. However, in doing so, there is significant risk that they will tend to exceed any estimated budgets or schedules for their projects - it being difficult to estimate what a process will do when you do not fully understand the process (what it is that you do) in the first place and cannot therefore control it or manage it effectively.

(c) Due to the lack of structure and formality, organizations at this level may over-commit, or abandon processes during a crisis, and it is unlikely that they will be able to repeat past successes. There tends to be limited planning, limited executive commitment or buy-in to projects, and limited acceptance of processes.

Level 2 - Repeatable

It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

Organizational implications

© Copy Right Virtual University of Pakistan

i) Processes and their outputs could be visible to management at defined points, but results may not always be consistent. For example, for project/program management processes, even though (say) some basic processes are established to track cost, schedule, and functionality, and if a degree of process discipline is in place to repeat earlier successes on projects with similar applications and scope, there could still be a significant risk of exceeding cost and time estimates.

Level 3 - Defined

It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization.

Organizational implications

(a) Process management starts to occur using defined documented processes, with mandatory process objectives, and ensures that these objectives are appropriately addressed.

Level 4 - Managed

It is characteristic of processes at this level that, using process metrics, management can effectively control the AS-IS process (e.g., for software development). In particular, management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Process Capability is established from this level.

Organizational implications

a) Quantitative quality goals tend to be set for process output - e.g., software or software maintenance.
b) Using quantitative/statistical techniques, process performance is measured and monitored and generally predictable and controllable also.

Level 5 - Optimizing

It is a characteristic of processes at this level that the focus is on continually improving process performance through both incremental and innovative technological changes/improvements.

Organizational implications

(a) Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement. Thus, process improvements to address common causes of process variation and measurably improve the organization’s processes are identified, evaluated, and deployed.

© Copy Right Virtual University of Pakistan

(b) The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives.
(c) Both the defined processes and the organization's set of standard processes are targets for measurable improvement activities.
(d) A critical distinction between maturity level 4 and maturity level 5 is the type of process variation addressed.

At maturity level 4, processes are concerned with addressing statistical *special causes* of process variation and providing statistical predictability of the results, and though processes may produce predictable results, the results may be insufficient to achieve the established objectives.
At maturity level 5, processes are concerned with addressing statistical *common causes* of process variation and changing the process (for example, shifting the mean of the process performance) to improve process performance. This would be done at the same time as maintaining the likelihood of achieving the established quantitative process-improvement objectives.

Extensions

Some versions of CMMI from SEI indicate a "level 0", characterized as "Incomplete". Some pundits leave this level out as redundant or unimportant, but Pressman and others make note of it.

Comparison of CMMI Representations Staged Continuous

What is a CMMI model representation? The answer requires an explanation of the structure of CMMI models. The basic building blocks in every CMMI model are called "process areas." A process area does not describe *how* an effective process is executed (e.g., entrance and exit criteria, roles of participants, resources). Instead, a process area describes *what* those using an effective process do (practices) and *why* they do those things (goals).

Selecting a Representation

When making the decision about which architectural representation to use for process improvement, comparative advantages of each approach as represented in the given Table.

Continuous Representation	**Staged Representation**
Grants explicit freedom to select the order of improvement that best meets the organization's business objectives. Enables increased visibility into the capability achieved within each individual process area. Supports a focus on risks specific to	Enables organizations to have a predefined and proven path. Builds on a relatively long history of use. Case studies and data exist that show return on investment. Permits comparisons across and among

individual process areas. Affords a more direct comparison of process improvement to ISO 15504 because the organization of process areas is derived from 15504. Allows the generic practices from higher capability levels to be more evenly and completely applied to all of the process areas.	organizations. Introduces a sequence of improvements, beginning with basic management practices and progressing through successive levels, each serving as a foundation for the next. Summarizes process-improvement results in a simple form—a single maturity-level number.

Project Management Concerns

Project management is the discipline of planning, organizing and managing resources to bring about the successful completion of specific project goals and objectives.

A project is a finite endeavor (having specific start and completion dates) undertaken to create a unique product or service which brings about beneficial change or added value. This finite characteristic of projects stands in sharp contrast to processes, or operations, which are permanent or semi-permanent functional work to repetitively produce the same product or service. In practice, the management of these two systems is often found to be quite different, and as such requires the development of distinct technical skills and the adoption of separate management.

There are several approaches that can be taken to managing project activities including agile, interactive, incremental, and phased approaches.

Regardless of the approach employed, careful consideration needs to be given to clarify surrounding project objectives, goals, and importantly, the roles and responsibilities of all participants and stakeholders.

The traditional approach

A traditional phased approach identifies a sequence of steps to be completed. In the "traditional approach", we can distinguish 5 components of a project (4 stages plus control) in the development of a project:



Typical development phases of a project

- Project initiation stage;
- Project planning or design stage;
- Project execution or production stage;
- Project monitoring and controlling systems;
- Project completion stage.

Not all the projects will visit every stage as projects can be terminated before they reach completion. Some projects probably don't have the planning and/or the monitoring. Some projects will go through steps 2, 3 and 4 multiple times.

Many industries utilize variations on these stages. For example, in bricks and mortar architectural design, projects typically progress through stages like Pre-Planning, Conceptual Design, Schematic Design, Design Development, Construction Drawings (or Contract Documents), and Construction Administration. In software development, this approach is often known as "waterfall development", i.e., one series of tasks after another in linear sequence. In software development many organizations have adapted the Rational Unified Process (RUP) to fit this methodology, although RUP does not require or explicitly recommend this practice. Waterfall development can work for small tightly defined projects, but for larger projects of undefined or unknowable scope, it is less suited. The Cone of Uncertainty explains some of this as the planning made on the initial phase of the project suffers from a high degree of uncertainty. This becomes specially true as software development is often the realization of a new or novel product, this method has been widely accepted as ineffective for software projects where requirements are largely unknowable up front and susceptible to change. While the names may differ from industry to industry, the actual stages typically follow common steps to problem solving — "defining the problem, weighing options, choosing a path, implementation and evaluation."

Software Quality Assurance

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or CMMI.

This definition emphasizes upon three important points:

- Software requirements are the foundation from which quality is measured. Lack of conformance is lack of quality
- Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
- A set of implicit requirements often goes unmentioned (ease of use, good maintainability etc.)

Another very important question is: Do you need to worry about it after the code has been generated? In fact, SQA is an umbrella activity that is applied throughout the software process.

© Copy Right Virtual University of Pakistan

Quality Assurance

Goal of quality assurance is to provide the management with the necessary data to be informed about product quality. It consists of auditing and reporting functions of management. If data provided through QA identifies problems, the management deploys the necessary resources to fix it and hence achieves desired quality control.

Cost of quality

A very significant question is: does quality assurance add any value. That is, is worth spending a lot of money in quality assurance practices? In order to understand the impact of quality assurance practices, we have to understand the cost of quality (or lack thereof) in a system.
Quality has a direct and indirect cost in the form of cost of prevention, appraisal, and failure.
If we try to prevent problems, obviously we will have to incur cost. This cost includes:

- Quality planning
- Formal technical reviews
- Test equipment
- Training

We will discuss these in more detail in the later sections.
The cost of appraisal includes activities to gain insight into the product condition. It involves in-process and inter-process inspection and testing.
And finally, failure cost. Failure cost has two components: internal failure cost and external failure cost. Internal failure cost requires rework, repair, and failure mode analysis. On the other hand, external failure cost involves cost for complaint resolution, product return and replacement, help-line support, warranty work, and law suits.

SQA Activities

There are two different groups involved in SQA related activities:

- Software engineers who do the technical work
- SQA group who is responsible for QA planning, oversight, record keeping, analysis, and reporting

Software engineers address quality by applying solid technical methods and measures, conducting formal and technical reviews, and performing well planned software testing. The SQA group assists the software team in achieving a high quality product.

SQA Group Activities

An SQA plan is developed for the project during project planning and is reviewed by all stake holders. The plan includes the identification of:

- Evaluations to be performed
- Audits and reviewed to be performed
- Standards that are applicable to the project
- Procedures for error reporting and tracking
- Documents to be produced by the SQA group

© Copy Right Virtual University of Pakistan

- Amount of feedback provided to the software project team

The group participates in the development of the project's software process description. The software team selects the process and SQA group reviews the process description for compliance with the organizational policies, internal software standards, externally imposed standards, and other parts of the software project plan.

Quality Control

The next question that we need to ask is, once we have defined how to assess quality, how are we going to make sure that our processes deliver the product with the desired quality. That is, how are we going to control the quality of the product?
The basic principle of quality control is to control the variation as variation control is the heart of quality control. It includes resource and time estimation, test coverage, variation in number of bugs, and variation in support.
From one project to another we want to minimize the predicted resources needed to complete a project and calendar time. This involves a series of inspection, reviews, and tests and includes feedback loop. So quality control is a combination of measurement and feedback and combination of automated tools and manual interaction.

The End

© Copy Right Virtual University of Pakistan