CS504 Current data final term paper
All current papers 2011 solved papers (eagle_eye)

1) Describe the the concept of cyclomatic complexity priefly (5)

Cyclomatic complexity is a software metric (measurement) Cyclomatic complexity is computed using a graph that describes the control flow of the program. The nodes of the graph correspond to the commands of a program.

Cyclomatic complexity is a quantitative measure of the logical complexity of a program. It defines number of independent paths in the basis set of a program. It provides an upper bound for the number of tests that must be conducted to ensure that all statements and branches have been executed at least once.

2) Give examples (atleast two) of business requirements (5)

For example, a business requirement to generate maximum revenue from the kiosk would imply the early implementation of features directly associated with selling more products or services to the customer, rather than glitzy features that appeal to only a subset of customers.

3) Write principles of unit testing (5)

- In unit testing, developers test their own code units (modules, classes, etc.) during implementation.
- Normal and boundary inputs against expected results are tested.
- Thus unit testing is a great way to test an API.

4) 5 general naming conventions in Java or C++ (5) **(P # 150)**

1. Names representing types must be nouns and written in mixed case starting with upper case.

2. Variable names must be in mixed case starting with lower case.

3. Names representing constants must be all uppercase using underscore to separate words.

4. Names representing methods and functions should be verbs and written in mixed case starting with lower case.

5. Names representing template types in C++ should be a single uppercase letter.

6. Global variables in C++ should always be referred to by using the :: operator.

7. Private class variables should have _ suffix.

8. Abbreviations and acronyms should not be uppercase when used as name.

9. Generic variables should have the same name as their type.

10. All names should be written in English.

12. The name of the object is implicit, and should be avoided in a method name.

```
line.getLength(); // NOT: line.getLineLength();
```

5) Difference between testing and inspection (3) (P # 210)

Inspection can be effective when same method can be used on a lot of different documents and testing is effective when it comes to rerunning the same test.

Inspections and testing are complementary and not opposing verification techniques. Both should be used during the verification and validation process. **Inspections** can check conformance with a specification but not conformance with the customer's real requirements **in testing,** one defect may mask another so several executions are required.

.

6) **Successful testing**

A successful testing is that which detect the defect (errors) and fixed them

7) Validation (P # 192)

- Does the product meet user expectations?
- Have you built the right product?

8) 3 rules for avoiding common mistakes

1.

array[i++] = i;

If i is initially 3, array[3] might be set to 3 or 4.

2.

array[i++] = array[i++] = x;

Due to side effects, multiple assignments become very dangerous. In this example, a whole depends upon when i is incremented.

3.

"," is very dangerous as it causes side effects. Let's look at the following statement:

int i, j = 0;

Because of the syntax, many people would assume that i is also being initialized to 0, while it is not. Combination of , and = -- is fatal. Look at the following statement:

a = b, c = 0;

A majority of the programmers would assume that all a, b, and c are being initialized to 0 while only c is initialized and a and b have garbage values in them. This kind of overlook causes major programming errors which are not caught easily and are caused only because there are side effects.

9) The following statement depicts which requirement engineering process?

"Constraints on the service function offered by the system such as timing constraints, constraints on the development process, standard etc"

10) How comments should be intended related to their position in code (example) (P # 176)

1. All comments should be written in English. In an international environment English is the preferred language.

2. Use // for all comments, including multi-line comments.

```
// Comment spanning
// more than one line
```

3. Comments should be indented relative to their position in the code.

11) What is meant by software debugging? (P # 213)

NEXT PAPER

Naming Convention 5 marks

It is a variable naming convention that includes information about the variable in its name (such as data type, whether it is a reference variable or a constant variable, etc). Every company and programmer seems to have their own flavor of Hungarian Notation. The advantage of Hungarian notation is that by just looking at the variable name, one gets all the information needed about that variable

Diff b/w test data and test cases 3 mark (P # 195)

Test Cases and Test Data

In order to test a software application, it is necessary to generate test cases and test data which is used in the application. **Test cases** correspond to application functionality such that the tester writes down steps which should be followed to achieve certain functionality. Thus a test case involves

- Input and output specification plus a statement of the function under test.
- Steps to perform the function
- Expected results that the software application produces

However, includes inputs that have been devised to test the system.

Exception 3mark (P # 37)

Exceptions:

1. The system will not allow a user to delete information that is being used system.

2. The system will not allow a user to delete another user that has subordinates.

Software validation 2mark (P # 192)

- Does the product meet user expectations?
- Have you built the right product?

Qualitative benefit 2mark (P # 207)

1) Assessment-oriented:

Writing the unit test forces us to deal with design issues - cohesion, coupling.

2) Confidence-building:

We know what works at an early stage. Also easier to change when it's easy to retest.

Software engineering tools for framework 3mark (P #13)

Any Engineering approach must be founded on organizational commitment to quality. That means the software development organization must have special focus on quality while performing the software engineering activities. Based on this commitment to quality by the organization,



Figure 2: A Software Engineering Framework

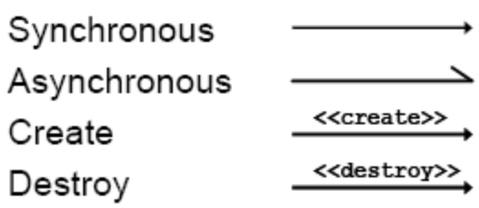
Diff b/w software and art factor 3mark

Collaboration diagram (P #111)

Collaboration diagrams can also be used to depict the dynamic behaviour of a system. They show how objects interact with respect to organizational units (boundaries!).

Types of messages in sequence diagram 2marks P #108)

Sequence diagrams can depict many different types of messages. These are: synchronous or simple, asynchronous, create, and destroy.



maintaing code guidline 5mark

14 FEB

40 mcqs

Define unit testing(2) P #207)

A software program is made up of units that include procedures, functions, classes etc. The unit testing process involves the developer in testing of these units. Unit testing is roughly equivalent to chip-level testing for hardware in which each chip is tested thoroughly after manufacturing. Similarly, unit testing is done to each module, in isolation, to verify its behaviors.

GUI in srs (2) P #62)

Adding user interface details in the SRS is controversial. The opponents of this argue that by adding GUI details to the SRS document, focus shifts from what to how – GUI is definitely part of the solution. On the other hand many people think that, it is still what not how and hence it should be made part of the SRS document. By adding the GUIs in the FS, requirements can be solidified with respect to scenario contents. It is my personal experience that the client appreciates more the contents of the SRS document if our SRS document contains the GUI details than if we don't have them there. This document is also going to be used as the base line for design, user manual, and test planning among other things. Presence of the UI details imply that these activities can start right after SRS is accepted and signed-off. Emergence of rapid GUI drafting tools has made the task a lot simpler than it used to be. Exploring potential user interfaces can be of help in refining the requirements and making the user-system interaction more tangible to both the user and the developer. User displays can help in project planning and estimation. A user interface might highlight weaknesses in addressing some of the non-functional requirements (such as usability), which are otherwise very hard to fix later on. If you cannot freeze the RS until UI is complete, requirement development process takes a longer time. However, we need to be very careful when we use GUIs to the SRS document. It is a very common mistake to use UI layouts as substitute of defining the functional requirements. We must remember that these are supplementary information and cannot replace other components of the SRS document. Any change in the requirements entails a change in the UI. If the requirements are not stable, this can mean a lot of rework.

self documenting code(2)s

A self documenting code is a code that explains itself without the n extraneous documentation, like flowcharts, UML diagrams, process-flow state diagrams, etc. That is, the meaning of the code should be evident just by reading the code without having to refer to information present outside this code.

3 rules of split lines (3) P #155)

- Break after a comma.
- Break after an operator.
- Align the new line with the beginning of the expression on the previous line.

how software differ from other engineering (3) P #41)

An important difference between software and another engineering discipline is that the software engineer has to work on problems that do not directly relate to software engineering. Whereas, an electrical engineer will work on electrical domain problems, a civil engineer will work on civil engineering problems and so on. So, software engineer has to learn user vocabulary and terms which they use in their routine operations.

define cohesion (3) P #72)

Cohesion implies that all parts of a component should have a close logical relationship with each other. That means, in the case some kind of change is required in the software, all the related pieces are found at one place.

- Cohesion is an internal property of a module.
- Cohesion describes the intra-component linkages
- Cohesion measures the independence of a module.

successful testing (3)

A successful testing is that which detect the maximum defect (errors) and fixed them.

5 objectives of testing (5) P #193)

Software testing objective

- The correct approach to testing a scientific theory is not to try to verify it, but to seek to refute the theory. That is to prove that it has errors. (Popper 1965)
- The goal of testing is to expose latent defects in a software system before it is put to use.
- A software tester tries to break the system. The objective is to show the presence of a defect not the absence of it.
- Testing cannot show the absence of a defect. It only increases your confidence in the software.
- This is because exhaustive testing of software is not possible – it is simply too expansive and needs virtually infinite resources.

Loop error bugs (5) P #220)

- Loop errors break down into several different subtypes.
- They occur around a loop construct in a program.
- Infinite loops, off-by-one loops, and improperly exited loops.

Why complex expression should be broken down (5) P #164)

Complex expressions should be broken down into multiple statements. An expression is considered to be complex if it uses many operators in a single statement.

DIFFERENCE BETWEEN REQUIREMENT STATEMENTS AND REQUIREMENT SPECIFICATIONS.) P #26)

Different levels of software requirements are documented in different documents. The two main documents produced during this phase are Requirement Statement and Requirement Specification. They are also called Requirement Definition and Functional Specification and are used to document user requirements and functional requirements respectively.

A good **Requirements statement** document must possess the following characteristics.
Complete - Each requirement must fully describe the functionality to be delivered.
Correct - Each requirement must accurately describe the functionality to be built.
Feasible - It must be possible to implement each requirement within the known capabilities and limitations of the system and its environment.

A good **Requirements specification** document should possess the following characteristics.
Complete - No requirement or necessary information should be missing.
Consistent – No requirement should conflict with other software or higher-level system or business requirements.
All programs must be written in Ada
The program must fit in the memory of the embedded micro-controller

11 FEB

Q1 Define modularity) P #72)_

Modularity is a tool that can help us in reducing the size of individual functions, making them more readable. One major advantage of modularity is that it allows the designer to apply the principle of separation of concern on individual modules

Q2 Define Unit testing P #207)

A software program is made up of units that include procedures, functions, classes etc. The unit testing process involves the developer in testing of these units. The unit test will establish some sort of artificial environment and then invoke routines in the module being tested.

Q3 What is the syntax used for naming objects in a sequence diagrams? P #107)_
The syntax used for naming objects in a sequence diagram is as follows:

- syntax: [instanceName][:className]

Q4 Define these terms: Branch Coverage, Statement Coverage. P #202)_

- **Statement Coverage**: In this scheme, statements of the code are tested for a successful test that checks all the statements lying on the path of a successful scenario.
- **Branch Coverage**: In this scheme, all the possible branches of decision structures are tested. Therefore, sequences of statements following a decision are tested.

Q5 Bit fields do suffer from a lack of portability b/w platforms.Why?P #183)

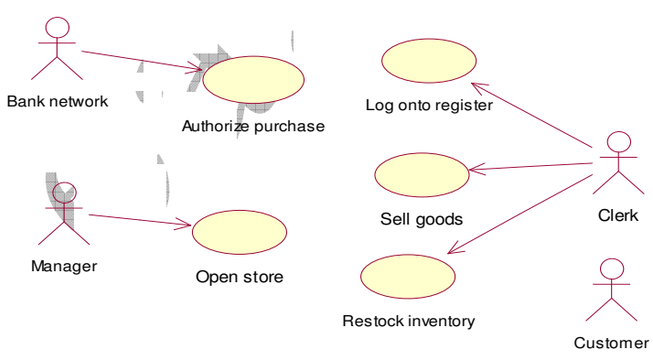
Bit fields are a convenient way to express many difficult operations. However, bit fields do suffer from a lack of portability between platforms:

- integers may be signed or unsigned
- Many compilers limit the maximum number of bits in the bit field to the size of an `integer` which may be either 16-bit or 32-bit varieties.
- Some bit field members are stored left to right others are stored right to left in memory.
- If bit fields too large, next bit field may be stored consecutively in memory (overlapping the boundary between memory locations) or in the next word of memory.

Bit fields therefore should not be used.

Q6 Consider the following Use Case diagram: P #33)

Identify the system actors in given use case diagram.



Q7 Why Special characters like TAB and page break must be avoided. P #33)

These characters are bound to cause problem for editors, printers, terminal emulators or debuggers when used in a multi-programmer, multi-platform environment.

Q8 Write down at least two (3) guideline that can avoid hazards caused by side effects. P #176)

1. never use “,” except for declaration

2. if you are initializing a variable at the time of declaration, do not declare another variable in the same statement

3. never use multiple assignments in the same statement

4. Be very careful when you use functions with side effects – functions that change the values of the parameters.

5. Try to avoid functions that

Q9 Discuss five points, how a “variable “is efficiently used in a program? P #159)

1. Variables should be initialized where they are declared and they should be declared in the smallest scope possible.

2. Variables must never have dual meaning. This enhances readability by ensuring all concepts are represented uniquely. Reduce chance of error by side effects.

3. Class variables should never be declared public. The concept of information hiding and encapsulation is violated by public variables. Use private variables and access functions instead. One exception to this rule is when the class is essentially a data structure, with no behavior (equivalent to a C++ `struct`). In this case it is appropriate to make the class' instance variables public.

4. Related variables of the same type can be declared in a common statement. Unrelated variables should not be declared in the same statement.

5. Variables should be kept alive for as short a time as possible. Keeping the operations on a variable within a small scope, it is easier to control the effects and side effects of the variable.

6. Global variables should not be used. Variables should be declared only within scope of their use. Same is recommended for global functions or file scope variables. It is easier to control the effects and side effects of the variables if used in limited scope.

Q10 Differentiate between Black box testing and white box testing. P #198)

Black box testing

In this type of testing, a component or system is treated as a black box and it is tested for the required behavior. This type of testing is not concerned with how the inputs are transformed into outputs.

white box testing

As opposed to black box testing, in structural or white box testing we look inside the system and evaluate what it consists of and how is it implemented. The inner of a system consists of design, structure of code and its documentation etc. Therefore, in white box testing we analyze these internal structures of the program and devise test cases that can test these structures.

Q 11What do we mean by ambiguous requirements. Explain with the help of a example. P #20)

Ambiguous requirements lead to ill-spent time and rework. Ambiguity means that two different readers of the same document interpret the requirement differently. Ambiguity arises from the use of natural language. Because of the imprecise nature of the language, different readers interpret the statements differently. As an example, consider the following Urdu Phrase: *"Rooko mut jan doo"*. Now, depending upon where a reader places the comma in this statement, two different readers may interpret it in totally different manner. If a comma is palced after *"Rooko"*, the sentence will become *"Rooko, mut jane doo"*, meaning *"don't let him go"*. On the other hand if the comma id placed after *"mut"*, the sentence will become *"Rooko mut, jane doo"*, meaning *"let him go"*. Ambiguous requirements therefore result in misunderstandings and mismatched expectations, resulting in a wasted time and effort and an undesirable product

Q12 You have learnt about Static analyzers, What are static analyzers? Give a checklist for static analysis. P #212)

Static analyzers are software tools for source text processing. They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the verification and validation team. These tools are very effective as an aid to inspections. But these are supplement to but not a replacement for inspections.

Checklist for static analysis

Data faults	• variable used before initialization • variable declared but never used • variables assigned twice but never used between assignments • possible array bound violations • undeclared variables
Control faults	• unreachable code • unconditional branches into loops
Input/Output faults	• variable output twice with no intervening assignment
Storage Management fault	• unassigned pointers • pointer arithmetic