NORTHERN INDIA ENGINEERING COLLEGE

MODEL TEST PAPER-1

Fourth semester [B.Tech]

Sub Code : **ETCS-202** Sub: **Software Engineering**

Note : ***Attempt Question no. 1 and any three from rest***

Q1(a) What are the strengths and weaknesses of DFD and Use Case Diagrams?

Ans Data Flow Diagrams :

Strength: Models the transformation of data, provides a framework for design/implementation

Weakness: easy to miss the relationship between data components; easy to fall into a design mentality

Use Case Diagrams:

Strength: Good for capturing the sequence of actions the system performs and the dialog with Actors; ideal for use in object modeling

Weakness: little or no connection to the data or the processes involved.

Q1 (b) What are the reasons for software crisis?

- Projects running over-budget.
- Projects running over-time.
- Software was very inefficient.
- Software was of low quality.
- Software often did not meet requirements.
- Projects were unmanageable and code difficult to maintain.
- Software was never delivered.

Q 1 (c): What are the different techniques to estimate size of program ?

Ans Two techniques to estimate size of program:-

1. LOC (Line Of Code) -- A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all line containing program header, declarations, executable and non-executable statements.
2. Function count – It measures the functionality from the user's point of view that is on the basis of what the user request and receives in return. Therefore it deals with the functionality being delivered and not with the line of code, source modules, files etc.

Q.1(d) What is the difference between Functional-oriented Approach and Object-oriented Design?

1.FOD: The basic abstractions, which are given to the user, are real world functions.
OOD: The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.

2.FOD: In this approach the state information is often represented in a centralized shared memory.
OOD: In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.

3.FOD approach is mainly used for computation sensitive application,
OOD: whereas OOD approach is mainly used for evolving system which mimicks a business process or business case.

4. FOD: Top down Approach
OOD: Bottom up approach

5. FOD: Begins by considering the use case diagrams and Scenarios.
OOD: Begins by identifying objects and classes.

Q1(e) What is risk exposure? What techniques can be used to control each risk?

Risk exposure is the product of probability of incurring a loss due to risk and the potential magnitude of that loss. The prioritization can be done in a quantitative way, by estimating the probability (0.1-1.0) and relative loss, on a scale of 1to 10. The higher the exposure the more aggressively the risk should be tackled.

Three techniques by which risk can be controlled:

1) Risk management planning: it produces a plan for dealing with each significant risk. It is useful to record decisions in the plan so that both customer and developer can review how problems area to be avoided as well as how they are to be handled when they arises.

2) Risk Monitoring: It is the process in which the project is in development progress periodically reevaluating the risk, their probability and likely impact .

3) Risk Resolution is the extension of the plans for dealing with each significant risk.

Q1 (f) What is software failure and how it differs from fault?

Software failure occurs when the software doesn't do what the user expects to see. The software fault (also known as software defect), on the other hand, is a hidden programming error. The software fault becomes a software failure only when the exact computation conditions are met, and the faulty portion of the code is executed on the CPU. The software fault can occur during normal usage. Other times the software fault occurs when the software is ported to a different hardware platform. Or when the software is ported to a different compiler. Or when the software gets extended.

Q1 (g) What are the different activities in the design framework of any software system development process?

Ans) The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. It consists a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

These activities are:

1. **System analysis, requirements definition:** defines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.
2. **Systems design:** describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudo-code and other documentation.
3. **Development:** The real code is written here.
4. **Integration and Testing:** brings all the pieces together into a special testing environment, then check for errors, bugs and interoperability.
5. **Acceptance, installation, deployment:** The final stage of initial development, where the software is put into production and runs actual business.
6. **Maintenance:** What happens during the rest of the software's life, i.e., changes, correction, additions, moves to a different computing platform and more. This is often the longest of the stages.

Q1 (h) Explain Alpha Testing and Beta Testing.

Ans ALPHA TESTING
The alpha testing is the first part of testing. The software needs to pass alpha testing, in order to move on to beta testing. If the software fails alpha testing, it will go in to redevelopment and be retested, until it passes. Typically, alpha and beta testing occur after the formal test plan has been successfully completed. Alpha is the first letter in the Greek alphabet, beta is the second.

Alpha testing is performed by the users within the organization developing the software. It is done in a lab environment so that user actions can be measured and analyzed. Its purpose is to measure real users' abilities to use and navigate the software before it can be released to the general public. Alpha testing includes unit testing, component testing, and system testing. The developers use either debugger software, or hardware-assisted debuggers, that help catch bugs in the system quickly.

BETA TESTING

Beta testing, generally involves a limited number of external users. At this time, beta test versions of software are distributed to a select group of external users, in order to give the program a real-world test. This is done to ensure that the product has few faults or bugs and that it can handle normal usage by its intended audience. Sometimes, the beta versions are made available to the open public to increase the feedback. If the audience finds any bugs or faults, they report it back to the developers, who then recreate the problem and fix it before the release. This process helps identify and mitigate defects that were missed during the formal test plan.

Q2 (a) Explain the significance of software reliability.

Ans **a) Computers are now cheap and fast:** There is little need to maximize equipment usage. Paradoxically, however, faster equipment leads to increasing expectations on the part of the user so efficiency considerations cannot be completely ignored.

b) Unreliable software is liable to be discarded by users: If a company attains a reputation for unreliability because of single unreliable product, it is likely to affect future sales of all of that company’s products.

c) System failure costs may be enormous: For some applications, such a reactor control system or an aircraft navigation system, the cost of system failure is orders of magnitude greater than the cost of the control system.

d) Unreliable systems are difficult to improve: It is usually possible to tune an inefficient system because most execution time is spent in small program sections. An unreliable system is more difficult to improve as unreliability tends to be distributed throughout the system.

e) Inefficiency is predictable: Programs take a long time to execute and users can adjust their work to take this into account. Unreliability, by contrast, usually surprises the user. Software that is unreliable can have hidden errors which can violate system and user data without warning and whose consequences are not immediately obvious. For example, a fault in a CAD program used to design aircraft might not be discovered until several plane crashers occurs.

f) Unreliable systems may cause information loss: Information is very expensive to collect and maintains; it may sometimes be worth more than the computer system on which it is processed. A great deal of effort and money is spent duplicating valuable data to guard against data corruption caused by unreliable software.

Q2. (b) What are the different CMM levels? What does CMM level specifies?

Ans. The CMM is used to judge the maturity of a software process. It has the following maturity levels.

- Initial (Maturity level 1) : There is no sound software egg. Management. It's on adhoc basis. Success of project rely on competent manager and good development team. However, usual pattern is time and cost overrun due to lack of management.
- Repeatable (Maturity level 2): At this level policies for managing a project and procedures to implement them are established. Planning and managing is based upon past experience with similar project. Regular measurement are done to identify problems and immediate action taken to prevent problem from becoming crisis.
- Defined(Maturity level 3): In this level sets of defined and documented standard processes are established and subject to some degree of improvement over time . These standard processes are in place and used to establish consistency of process performance across the organization.
- Managed (Maturity level 4): It's the characteristic of process at this level that using process metrics, management can effectively control for software development. In particular Management can identify ways to adjust and adapt the process to particular projects without measurable loses of quality or deviation from specifications. Process capability is established from this level.
- Optimizing(Maturity level 5): In this level the focus is on continually improving process performance through both incremental and innovative technological changes improvement.

Q3(a) Explain all the levels of COCOMO Model.

The **Constructive Cost Model** (**COCOMO**) is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current as well as future project characteristics.

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, *Basic COCOMO* is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (*Cost Drivers*). *Intermediate COCOMO* takes these Cost Drivers into account and *Detailed COCOMO* additionally accounts for the influence of individual project phases.

Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC), (KLOC).

COCOMO applies to three classes of software projects:

- Organic projects - "small" teams with "good" experience working with "less than rigid" requirements

- Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements
- Embedded projects - developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects.(hardware, software, operational, ...)

The basic COCOMO equations take the form

Effort Applied (E) = $a_b(\text{KLOC})^{b_b}$ **[person-months]**

Development Time (D) = $c_b(\text{Effort Applied})^{d_b}$ **[months]**

People required (P) = Effort Applied / Development Time **[count]**

where, **KLOC** is the estimated number of delivered lines (expressed in thousands) of code for project. The coefficients a_b, b_b, c_b and d_b are given in the following table:

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Basic COCOMO is good for quick estimate of software costs. However it does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.

Intermediate COCOMO

Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers",each with a number of subsidiary attributes:-

- Product attributes
  - Required software reliability
  - Size of application database
  - Complexity of the product
- Hardware attributes

    - Run-time performance constraints
    - Memory constraints
    - Volatility of the virtual machine environment
    - Required turnabout time
- Personnel attributes
    - Analyst capability
    - Software engineering capability
    - Applications experience
    - Virtual machine experience
    - Programming language experience
- Project attributes
    - Use of software tools
    - Application of software engineering methods
    - Required development schedule

The Intermediate Cocomo formula now takes the form:

$$\mathbf{E=a_i(KLoC)^{(b_i)}.EAF}$$

where E is the effort applied in person-months, **KLoC** is the estimated number of thousands of delivered lines of code for the project, and **EAF** is the factor calculated above. The coefficient $\mathbf{a_i}$ and the exponent $\mathbf{b_i}$ are given in the next table.

Software project	$\mathbf{a_i}$	$\mathbf{b_i}$
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

The Development time **D** calculation uses **E** in the same way as in the Basic COCOMO.

Detailed COCOMO

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

The detailed model uses different effort multipliers for each cost driver attribute. These **Phase Sensitive** effort multipliers are each to determine the amount of effort required to complete each phase. In detailed cocomo,the whole software is divided in different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort

In detailed COCOMO, the effort is calculated as function of program size and a set of cost drivers given according to each phase of software life cycle.

A Detailed project schedule is never static.

The five phases of detailed COCOMO are:-

- plan and requirement.
- system design.
- detailed design.
- module code and test.
- integration and test.

Q3(b) Determine the effort required to develop the software product and nominal development time assuming the size of an organic software product has been estimated to be 25K times of code.

Ans. $E=a_b(KLOC)^{b}{}_{b}$

$E=2.4(25)^{1.05}$

$=2.4*29.36=70.46=70PM$

$D=c_b(E)^{d}{}_{b}$

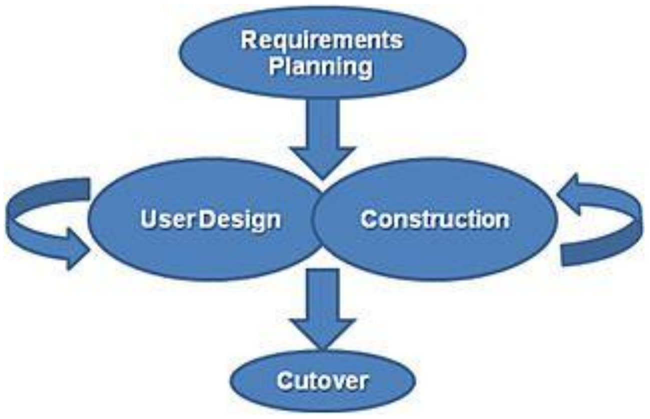
$=2.5(70)^{0.38}$

$=2.5(50.2)$

$=12.55M$

Q4(a) Explain Rapid Application Development Model. what are the advantages and disadvantages of this model?

Ans RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.



Four phases of RAD

1. **Requirements Planning phase** – combines elements of the system planning and systems analysis phases of the Systems Development Life Cycle (SDLC). Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.
2. **User design phase** – during this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. *User Design* is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.
3. **Construction phase** – focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.
4. **Cutover phase** – resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.

Advantages of the RAD model:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated codegeneration is very high.

Q4(b) What are the characteristics of a good SRS?

Ans

1. Complete
A complete requirements specification must precisely define all the real world situations that will be encountered and the capability's responses to them. It must not include situations that will not be encountered or unnecessary capability features.

2. Consistent
System functions and performance level must be compatible and the required quality features (reliability, safety, security, etc.) must not contradict the utility of the system.

3. Correct
The specification must define the desired capability's real world operational environment, its interface to that environment and its interaction with that environment. It is the real world aspect of requirements that is the major source of difficulty in achieving specification correctness.

4. Modifiable

Related concerns must be grouped together and unrelated concerns must be separated. Requirements document must have a logical structure to be modifiable.

5. Ranked
Ranking specification statements according to stability and/or importance is established in the requirements document's organization and structure. The larger and more complex the

problem addressed by the requirements specification, the more difficult the task is to design a document that aids rather than inhibits understanding.

6. Testable
A requirement specification must be stated in such as manner that one can test it against pass/fail or quantitative assessment criteria, all derived from the specification itself and/or referenced information. Requiring that a system must be "easy" to use is subjective and therefore is not testable.

7.Traceable
Each requirement stated within the SRS document must be uniquely identified to achieve traceability. Uniqueness is facilitated by the use of a consistent and logical scheme for assigning identification to each specification statement within the requirements document.

8.Unambiguous
A statement of a requirement is unambiguous if it can only be interpreted one way. This perhaps, is the most difficult attribute to achieve using natural language. The use of weak phrases or poor sentence structure will open the specification statement to misunderstandings.

9.Valid
To validate a requirements specification all the project participants, managers, engineers and customer representatives, must be able to understand, analyze and accept or approve it. This is the primary reason that most specifications are expressed in natural language.

10.Verifiable
In order to be verifiable, requirement specifications at one level of abstraction must be consistent with those at another level of abstraction. Most, if not all, of these attributes are subjective and a conclusive assessment of the quality of a requirements specification requires review and analysis by technical and operational experts in the domain addressed by the requirements.

Q5 Write short notes on following:

a) DATA FLOW TESTING:

Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects. Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

Advantages of Data Flow Testing:

Data Flow testing helps us to pinpoint any of the following issues:

- A variable that is declared but never used within the program.
- A variable that is used but never declared.
- A variable that is defined multiple times before it is used.
- Deallocating a variable before it is used.

b) REVERSE ENGINNERING

Reverse engineering is the process of discovering the technological principles of a device, object, or system through analysis of its structure, function, and operation.[1] It often involves disassembling something (a mechanical device, electronic component, computer program, or biological, chemical, or organic matter) and analyzing its components and workings in detail, just to re-create it. Reverse engineering is done for maintenance or to create a new device or program that does the same thing, without using original, or simply to duplicate it.

Reverse engineering has its origins in the analysis of hardware for commercial or military advantage.[2] The purpose is to deduce design decisions from end products with little or no additional knowledge about the procedures involved in the original production. The same techniques are subsequently being researched for application to legacy software systems, not for industrial or defence ends, but rather to replace incorrect, incomplete, or otherwise unavailable documentation.

c) CONFIGURATION MANAGEMENT

Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional and physical attributes with its requirements, design and operational information throughout its life CM, when applied over the life cycle of a system, provides visibility and control of its performance, functional and physical attributes. CM verifies that a system performs as intended, and is identified and documented in sufficient detail to support its projected life cycle. The CM process facilitates orderly management of system information and system changes for such beneficial purposes as to revise capability; improve performance, reliability, or maintainability; extend life; reduce cost; reduce risk and liability; or correct defects. The relatively minimal cost of implementing CM is returned many fold in cost avoidance. The lack of CM, or its ineffectual implementation, can be very expensive and sometimes can have such catastrophic consequences such as failure of equipment or loss of life