B.E.

Sixth Semester Examination, 2010

Principles of Software Engineering (CSE-302-E)

Note : Attempt any **five** questions. All questions carry equal marks.

Q. 1. (a) Explain Iterative enhancement model of software development life-cycle.

Ans. Iterative Enhancement Model : Iterative enhancement model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but there may be conducted in several cycles. A usable product is released at the end of the each cycle, with each release providing additional functionality. Increment process models are effective in the situations where requirement are defined precisely and there is no confusion about the functionality of the final product. (Although, functionality can be delivered in phases as per desired priorities). After every cycle, a usable product is given to the customer. For example, in the university automation software library automation module may be delivered in the first phase and examination automation module in the second phase and as so on. Every new cycle will have an additional functionality. Increment process models are popular particularly when we have to quickly deliver a limited functionality system.

During the first requirements analysis phase, customers and developers specify as many requirements as possible and prepare a SRS document. Developers and customers then prioritize these requirements. Developers implement the specified requirements in one or more cycle of design, implementation and test based on the defined priorities. The model is given in figure shown below.

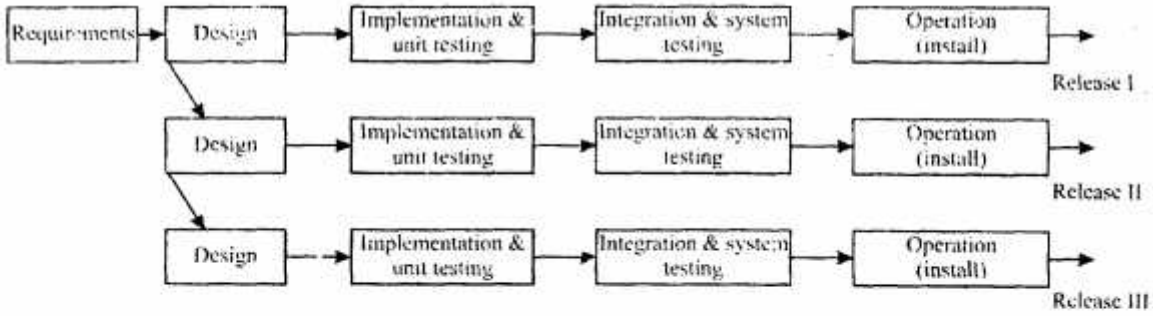


Fig. Iterative enhancement model

The aim of the waterfall and prototyping models is the delivery of a complete, operational and good quality product. In contrast, this model does deliver an operational quality product at each release, but one that satisfies only a subset of the customer's requirements. The complete product is divided into releases, and the developer delivers the product release by release. A typical product will usually have many releases as shown in figure. At each release, customer has an operational quality product that does a portion of what is required. The customer is able to do some useful work after first release. With this model, first release may be available within few weeks or months, whereas the customer generally waits months or years to receive a product using the waterfall and prototyping model.

Q. 1. (b) Discuss the parameters for the selection of a life cycle mode.

Ans. Parameters for the Selection of a Life-Cycle Model :

The selection of a suitable model is based on the following characteristics/categories/parameters :

(i) Requirements (ii) Development team

(iii) Users (iv) Project types and associated risk

(i) Characteristics of Requirements : Requirement are very important for the selection of an appropriate model. There are number of situations and problems during requirements capturing and analysis. The details are given below in table :

Selection of a Model based on Characteristics of Requirements

Requirements	Water-fall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Are requirements easily understandable and defined ?	Yes	No	No	No	No	Yes
Do we change require-ments quite often ?	No	Yes	No	No	Yes	No
Can we define requirements early in the cycle ?	Yes	No	Yes	Yes	No	Yes
Requirements are indicating a complex system to be built.	No	Yes	Yes	Yes	Yes	No

(ii) Status of Development Team : The status of development team in terms of availability, effectiveness, knowledge, intelligence, team work etc.. is very important for the success of the project. If we know above mentioned parameters and characteristics of the team, then we may choose an appropriate life-cycle model for the project. Some of the details are given below :

Selection based on Status of Development Team

Development Team	Water-fall	Prototype	Iterative enhancement	Evoluationary development	Spiral	RAD
Less experience on similar projects	No	Yes	No	No	Yes	No
Less domain knowledge (new to the technology)	Yes	No	Yes	Yes	Yes	No
Less experience on tools to be used	Yes	No	No	No	Yes	No
Availability of training, if required	No	No	Yes	Yes	No	Yes

(iii) Involvement of Users : Involvement of users increases the understandability of the project. Hence user participation, if available, plays a very significant role in the selection of an appropriate life cycle model.

Some issues are discussed below :

Selection Based on User's Participation

Involvement of Users	Water-fall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
User involvement in all phases.	No	Yes	No	No	No	Yes
Limited user participation	Yes	No	Yes	Yes	Yes	No
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No
Users are experts of problem domain	No	Yes	Yes			

(iv) Type of Project and Associated Risk :

Very few models incorporate risk assessment. Project type is also important for the selection of model. Some issues are discussed below :

Selection Based on Type of Project with Associated Risk

Project Type and Risk	Water-fall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Project is the enhancement of the existing system	No	No	Yes	Yes	No	Yes
Funding is stable for the project	Yes	Yes	No	No	No	Yes
High reliability requirements	No	No	Yes	Yes	Yes	No
Tight project schedule	No	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes
Are resources (time, money, people etc.) scarce ?	No	Yes	No	No	Yes	No

An appropriate model may be selected based on options given in four tables. Firstly, we have to answer the questions presented for each category by circling a Yes or No in each table. Rank the importance of each category, or question within the category in terms of the project for which we want to select a model. The total number of circled responses for each column in the tables decide an appropriate model.

Q. 2. (a) How do you calculate function points using FPA ? Explain with an example.

Ans. The five functional units are ranked according to their complexity i.e., low, average, or high using a set of prescriptive standards organizations that use Function Point (FP) methods develop criteria for determining whether a particular entry is low, average or high.

After classifying each of the five function types, the Unadjusted Function Points (UFP) are calculated using predefined weights for each function type as given below.

Table 1 : Functional Units with Weighting Factors

Functional Units	Weighting Factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal Logical Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

Table 2 : UFP Calculation Table

Functional Units	Count Complexity	Complexity Totals	Functional Unit Totals
External Inputs (EIs)	☐ Low × 3 = ☐		
	☐ Average × 4 = ☐		
	☐ High × 6 = ☐		☐
External Outputs (EOs)	☐ Low × 4 = ☐		
	☐ Average × 5 = ☐		
	☐ High × 7 = ☐		☐
External Inquiries (EQs)	☐ Low × 3 = ☐		
	☐ Average × 4 = ☐		
	☐ High × 6 = ☐		☐

Internal Logical Files (ILFs)	☐ Low × 2 = ☐ ☐ Average × 10 = ☐ ☐ High × 15 = ☐	☐
External Interface Files (EIFs)	☐ Low × 5 = ☐ ☐ Average × 7 = ☐ ☐ High × 10 = ☐	☐
Total Unadjusted Function Point Count		☐

The procedure for the calculation of UFP in mathematical form is given below :

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} j_{ij} W_{ij}$$

Where i indicates the row and j indicates the column of table 1.

W_{ij} : It is the entry of the ith row and jth column of table 1.

Z_{ij} : It is the count of the number of functional units of type i that have been classified as having the complexity corresponding the column j.

$$FP = UFP * CAF.$$

Where CAF is complexity adjustment factor and is equal to $[0.65 + 0.01\lambda. \Sigma F_i]$. The F_i (i = 1 to 14) are the degrees of influence and are based on response to questions.

Example :

Consider a project with following functional units :

Number of user inputs = 50
Number of user outputs = 40
Number of user enquiries = 35
Number of user files = 06
Number of external interfaces = 04

Assume all complexity adjustment factors, weighting factors are average compute the function points for the project.

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} W_{ij}$$

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7$$
$$= 200 + 200 + 140 + 60 + 28 = 628$$

$$CAF = (0.65 + 0.01 \Sigma F_i)$$
$$= (0.65 + 0.01 (14 \times 3))$$
$$= 0.65 + 0.42 = 1.07$$

$$FP = UFP \times CAF$$
$$= 628 \times 1.07 = 672$$

Q. 2. (b) Explain one model for estimating the cost of a software.

Ans. The Constructive Cost Model (COCOMO) : This model gained rapid popularity following the publication of B.W. Boehm's excellent book Software Engineering Economics in 1981. COCOMO is a hierarchy of software cost estimation models, which include basic, intermediate and detailed sub models.

The model aims at estimating, in a quick and rough fashion, most of the small to medium sized software projects. Three modes of software development are considered in this mode : organic, semi-detached and embedded. In the organic mode, a small team of experienced developers develops software in a very familiar environment. The size of the software development in this mode ranges from small (a few KLOC) to medium (a few tens of KLOG), while in other two modes the size ranges from small to very large (a few hundreds of KLOC).

In the embedded mode of software development, the project has tight constraints, which might be related to the target processor and its interface with the associated hardware. The problem to be solved is unique and so it is often hard to find experienced persons, as the same does not usually exist.

The semi-detached mode is an intermediate mode between the organic mode and embedded mode. The comparison of all there modes is given in table below :

The Comparison of Three COCOMO Modes

Mode	Project-Size	Nature of Project	Development Environment
Organic	Typically 2—50 KLOC	Small size project, experienced developers in the familiar environment.	Familiar & In house
Semi-detached	Typically 50—300 KLOC	Medium size project, Medium size team, Average previous experience on similar projects.	Medium
Embedded	Typically over 300 KLOC	Large project, real time systems, complex interfaces, very little previous experience.	Complex Hardware/ customer Interfaces required.

The basic COCOMO equations take the form

$$E = a_b \text{ (KLOC)}^{b_b}$$

$$D = c_b (E)^{d_b}$$

Where E is effort applied in person-months, and D is the development time in months. The coefficients a_b, b_b, c_b and d_b are given in table below :

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

When effort and development time are known, the average staff size to complete the project may be calculated as :

$$\text{Average staff size (SS)} = \frac{E}{D} \text{ persons}$$

When project size is known, the productivity level may be calculated as :

$$\text{Productivity (P)} = \frac{\text{KLOC}}{E} \text{ KLOC/PM.}$$

With the basic model, the software estimator has a useful tool for estimating quickly, by two runs on a pocket calculator, the cost and development time of a software project, once the size is estimated.

Q. 3. (a) What are the key concepts in designing a software ?

Ans. Object oriented design is not dependent on any specific implementation language. Problems are modelled using objects. Objects have :

(i) Behaviour (they do things).

(ii) State (which changes when they do things).

The various terms related to object oriented design are objects, classes, abstraction, inheritance and polymorphism.

(i) Objects : The word "object" is used very frequently and conveys different meaning in different circumstances. Here, meaning is an entity able to save a state and which offers a number of operations to either examine or affect this state. All objects have unique identification and are distinguishable.

(ii) Messages : Conceptually, objects communicate by message passing. Message consist of identity of the target object, the name of the requested operation and any other operation needed to perform the function. Messages are often implemented as procedure or function calls (name = procedure name, information = parameter list).

(iii) Abstraction : In object oriented design, complexity is managed using abstraction. Abstraction is the elimination of the irrelevant and amplification of the essentials.

(iv) Class : In any system, there shall be number of objects. Some of the objects may have common characteristics and we can group the objects according to these characteristics. This type of grouping is known as a class. Hence, a class is a set of objects that share a common structure and a common behaviour. Classes are useful because they act as a blue print for objects.

(v) Attributes : An attribute is a data value held by the objects in a class. The square class has two attributes : a colour and arrays of points. Each attribute has a value for each object, instance.

(vi) Operations : An operation is a function or transformation that may be applied to or by objects in a class. The behaviour of the operation depends on the class of its target.

(vii) Inheritance : The low level classes (known as subclasses or derived classes) inherit state and behaviour from this high level class (known as a super class or base class).

(viii) Polymorphism : When we abstract just the interface of an operation and leave the implementation to subclasses it is called a polymorphic operation and process is called polymorphism.

(ix) Encapsulation (Information Hiding) : Encapsulation is also commonly referred to as "Information Hiding." It consists of the separation of the external aspects of an object from the internal implementation details of the object. The external aspects of an object are accessible by other objects.

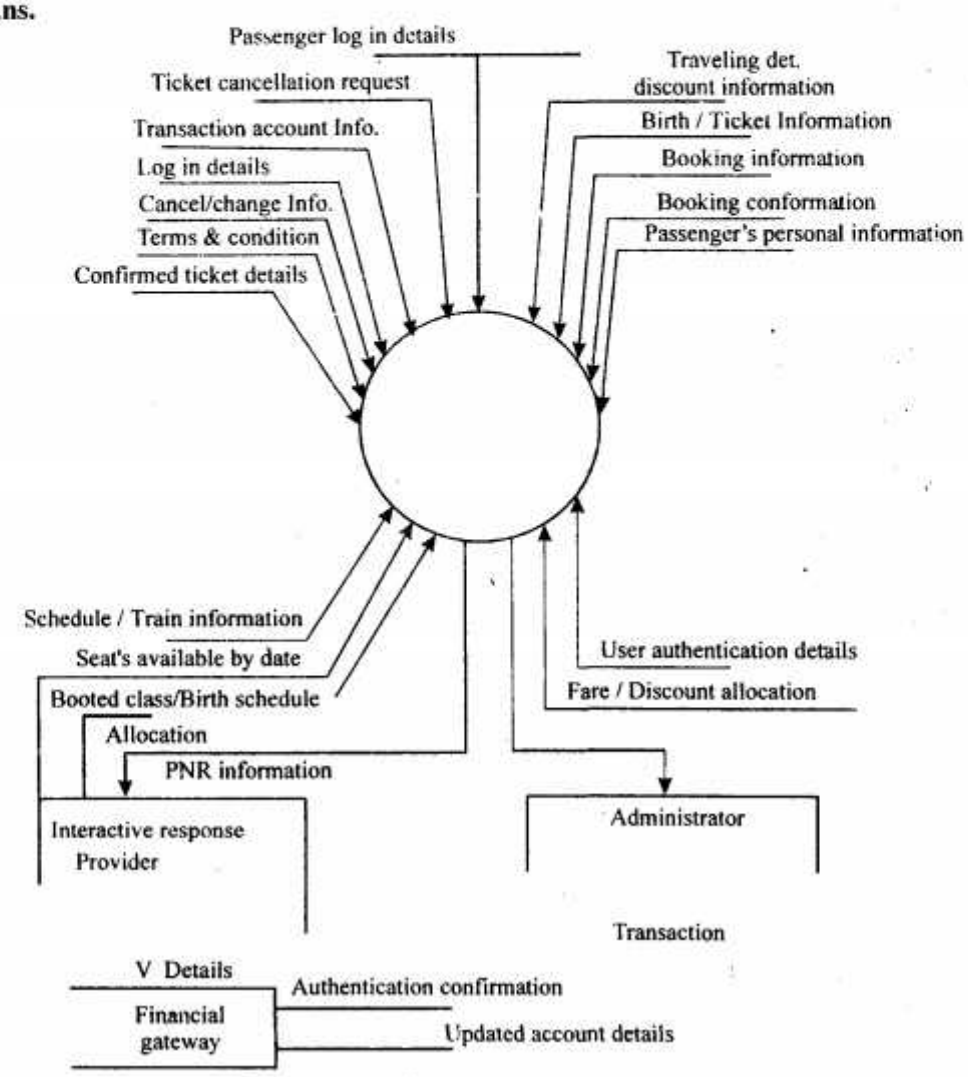
through methods of object, while the internal implementation of those methods are hidden from the external object sending the message.

Encapsulation deals with permitting or restricting a client. Class abiiity to modify the attributes or invoke the methods of the class or object of concern. Thus, encapsulation protects (a) an objects's internal, state from being corrupted by its clients and (b) client code from changes in the object's implementation.

(x) Hierarchy : Hierarchy involves organizing something according to some particular order or rank (e.g., complexity, responsibility etc.). It is another mechanism for reducing the complexity of software by being able to treat and express sub-types in a generic ways. This hierarchy is implemented in software via a mechanism called "Inheritance."

Q. 3. (b) Draw level-1 DFD for Railway Reservation system.

Ans.



Q. 4. (a) What are different risk management activities ?

Ans. Risk management is the identification, assessment and prioritization of risks followed by coordinated and economical application of resources to minimize, monitor, and control the probability and/or impact of unfortunate events or to maximize the realization of opportunities.

Principles of Risk Management :

Risk management should :

(i) create value

(ii) be an integral part of organizational process

(iii) be part of decision making

(iv) be systematic and structured.

Process :

"Risk Management ..... Principles and guidelines on implementation," the process of risk management consists of several steps as follows :

Establishing the Context : Establishing the context involves :

(i) Identification of risk in a selected domain of interest.

(ii) Planning the remainder of the process.

(iii) Mapping out the following :

(i) The social scope of risk management.

(ii) The identity and objective of stakeholders.

Identification : After establishing the context, the next step in the process of managing risk is to identify potential risks.

The chosen method of identifying risks may depend on culture, industry practice and compliance. Common risk identification methods are :

(i) Objectives-based Risk Identification : Organizations and project teams have objectives.

(ii) Scenario-based Risk Identification : In this different scenarios are created. The scenarios may be the alternative ways to achieve an objective.

(iii) Common-risk Checking : In several industries, lest with known risks are available.

Assessment : Once risk have been identified, they must then be assessed as to their potential severity of loss and to the probability of occurrence. The fundamental difficulty in risk assessment is determining the rate of occurrence since statistical information is not available on all kinds of past incidents.

Potential Risk Treatments : Once risks have been identified and assessed, all techniques to manage the risk fall into one or more of these four major categories :

(i) Avoidance : This include not performing an activity that could carry risk.

(ii) Reduction : Risk reduction or "optimisation" involves reducing the severity of the loss or the likelihood of the loss from occuring.

(iii) Sharing : It defined as sharing with another party the burden of loss or the benefit of gain, from a risk, and the measure to reduce a risk.

(iv) Retention : Involves accepting the loss or benefit of gain, from a risk when it occurs.

Create a Risk Management Plan : Select appropriate controls or counter-measures to measure each risk. The risk management plan should propose applicable and effective security controls for managing the risk.

Implementation : Implementation follows all of the planned methods for mitigating the effect of the risks. Purchase insurance policies for the risks that have been decided to be transferred to an insurer, avoid all risks that can be avoided without sacrificing the entity's goals, reduce others and retain the rest.

Q. 4. (b) What is software quality ? Discuss the software quality attributes.

Ans. Software Quality : Different people understand different meanings of quality like :

(i) Conformance to requirements

(ii) Fitness for the purpose

(iii) Level of satisfaction

If a product is meeting its requirement, we may say it is good quality product. Quality has many characteristics and some are related to each other.

In software, the quality is commonly recognised as "lack of bugs" in the program. If a software has too many functional defects, then, it is not meeting its basic requirement of functionality. This is usually expressed in two ways :

(i) Defect Rate : Number of defects per million lines of source code, per function point or any other unit.

(ii) Reliability : Generally measured as number of failures per '*t*' hours of operation, mean time to failure or probability of failure free operation in a specified time under specified environment.

When we deal with software quality, a list of attributes is required to be defined that are appropriate for software.

The details of software quality attributes are given below :

(i) Reliability : The extent to which a software performs its intended functions without failure.

(ii) Correctness : The extent to which a software meets its specifications.

(iii) Consistency and Precision : The extent to which a software is consistent and given result with precision.

(iv) Robustness : The extent to which a software tolerates the unexpected problems.

(v) Simplicity : The extent to which a software is simple in its operations.

(vi) Traceability : The extent to which an error is traceable in order to fix it.

(vii) Usability : The extent of effort required to learn, operate and understand the functions of the software.

(viii) Accuracy : Meeting specification with precision.

(ix) Clarity and Accuracy of Documentation : The extent to which documents are clearly and accurately written.

(x) Conformity of Operation Environment : The extent to which a software is in conformity of operational environment.

(xi) Completeness : The extent to which a software has specified functions.

(xii) Efficiency : The amount of computing resources and code required by software to perform a function.

(xiii) Testability : The effort required to test a software to ensure that it performs its intended functions.

(xiv) Maintainability : The effort required to locate and fix an error during maintenance phase.

(xv) Deadability : The extent to which a software is readable in order to understand.

(xvi) Adaptability : The extent to which a software as adaptable to new platforms and technology.

(xvii) Expandability : The extent to which a software is expandable without undesirable side effects.

(xviii) Portability : The effort required to transfer a program from one platform to another platform.

Q. 5. (a) Define the following terms :

Error, Bug, Fault, Defect, Failure, Test case, Test suite.

Ans. (i) Error : The word error entails different meaning. The concrete meaning of the Latin word error is ''wondering'' or ''stroying''. This may be syntax error or misunderstanding of specification.

(ii) Bug : When developer make mistakes during coding, we call these mistakes as bugs. Most bugs arise from mistakes and errors made by people in either a program's source code or its design.

(iii) Fault : An error may lead to one or more faults. It is more precise to say that a fault is the representation of an error. Where representation is the mode of expression, such as narrative text, dfd, FR diagram etc.

(iv) Defect : The lack of something necessary or desirable for completion or perfection. An imperfection that causes inadequacy or failure.

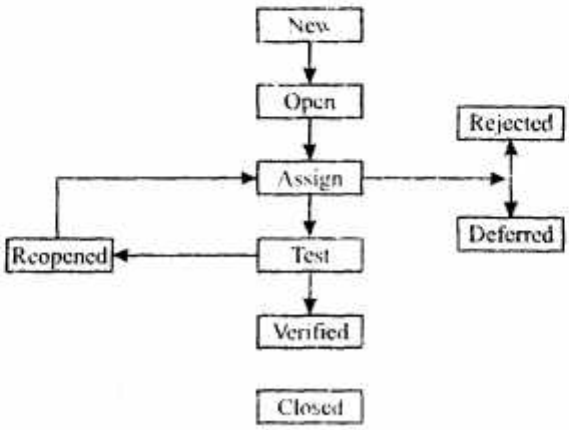
(v) Failure : A failure occur when a fault executes. It is the departure of the output program from the expected output. Hence failure is dynamic. A fault may lead to many failures.

(vi) Test Cases : Test cases describes an input description and an expected output description.

(vii) Test Suite : The set of test cases is called a test suite. We may have a test suite of all possible test cases.

Q. 5. (b) Explain life cycle of a Bug.

Ans. Bug Life Cycle : In software development process, the bug has a life-cycle. The bug should go through the life-cycle to be closed. A specific life-cycle ensures that the process is standardized. The bug attains different states in the life cycle. The life cycle of the bug can be shown diagrammatically as follows :



(i) New : When the bug is posted for the first time, its state will be ''New''.

(ii) Open : After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as ''OPEN''.

(iii) Assign : Once the lead changes the state as "Open", he assigns the bug to corresponding developer or developer team.

(iv) Test : Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. It specifies that the bug has been fixed and is released to testing team.

(v) Deferred : The bug, changed to deferred state means the bug is expected to be fixed in next releases. Some of them are priority of the bug may be low, lack of time for the release.

(vi) Rejected : If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "Rejected".

(vii) Duplicate : If the bug is reaped twice or the two bugs mention the same concept of the bug, then one bug status is changed to "Duplicate".

(viii) Verified : If the bug is not present in the software, he approves that the bug is fixed and changes the status to "verified."

(ix) Reopened : If the bug still exists even after the bug is fixed by the developer, the tester changes the stapes to "Reopened."

(x) Closed : If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "closed".

Q. 5. (c) What is Software Testing ? What are various testing principles ?

Ans. Software Testing : It is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.

Software testing can also be stated as the process of validating and verifying that a software product :

(i) meets the business and technical requirements that guided its design and development.

(ii) works as expected; and

(iii) can be implemented with the same characteristics.

Principles of Software Testing :

Software Testing :

(i) Test is a Formal Activity : It involves a strategy and a systematic approach. Tests are always specified and recorded.

(ii) Test is a Planned Activity : The workflow and the expected results are specified. Therefore the duration of the activities can be estimated. The point in time where tests are executed is defined.

(iii) Test is the formal proof of software quality.

Overview of Test Methods :

(i) Static : The software is not executed but analyzed offline. In this category could be code inspections, lint checks etc.

(ii) Dynamic : This requires the execution of the software or parts of the software. It can be executed in the target system.

(iii) Structural : There are so called "white-box tests" because they are performed with the knowledge of the source code details.

(iv) Functional : There are the so called "black box" tests. The software is regarded as a unit with unknown content.

Test by Progressive Stages :

(i) Module : A module is the smallest unit of source code. It is too small to allow functional tests. However it is ideal candidate for white-box tests.

(ii) Component : This is the black box test of modules or groups of modules which represent certain functionality. Components can be step by step integrated to bigger components and tested as such.

(iii) Integration : The integration depends on the kind of system. The integration is still done in the laboratory.

(iv) System : This is a black-box test of the complete software in the target system. The environmental conditions have to be realistic complete original hardware in the destination.

Q. 6. (a) What are different levels of Testing ?

Ans. Levels of Testing : Our emphasis during testing is to examine and modify the source code. There are three levels of testing i.e., individual module to the entire software system.

(i) Unit Testing : Unit testing is the process of taking a module and running it in isolation from the rest of the software product by using prepared test cases and comparing the actual results with the results predicted by the specification and design of the module. One purpose of testing is to find (and remove) as many errors in the software as practical. There are number of reasons in support of unit testing than testing the entire product.

(i) The size of a single module is small enough that we can locate an error fairly easily.

(ii) The module is small enough that we can attempt to test it in some demonstrably exhaustive fashion.

(iii) Confusing interactions of multiple errors in widely different parts of the software are eliminated.

(ii) Integration Testing : The purpose of unit testing is to determine that each independent module is correctly implemented. This gives little chance to determine that the interface between modules is also correct, and for this reason, integration testing must be performed. One specific target of integration testing is the interface. Whether parameters match on both sides as to type, permissible ranges, meaning and utilization. With integration testing, we move slowly away from structural testing and toward functional testing, which treats a module as an impenetrable mechanism for performing a function. As the aggregated modules become larger and larger, we lose our ability to think about path coverage and domains, and must be satisfied with simply determining that the product seems to do what we intended.

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new input/output may occur, and new control logic is invoked.

(iii) System Testing : Of the three levels of testing, the system level is closet to everyday experience.

As we know, software is one component of a large computer based system. Ultimately, software is incorporated with other system components (e.g., new hardware, information), and thus a series of special tests are to be conducted. Many times, software product are designed to run on a variety of hardware configurations. The software should actually be tested on many different hardware set-ups, although the full range of memory, processor, operating system and peripheral possibilities may be too large for complete testing. There are many types of specifications and we should be aware of those as we perform system testing.

During system testing, we should evaluate a number of attributes of the software that are vital to the user. These attributes represent to the operational correctness of the product and may be part of the software specification.

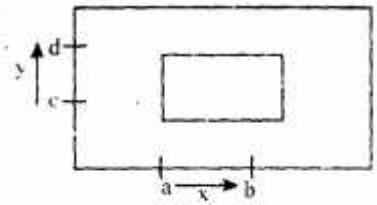
Q. 6. (b) Design various test cases to find out the roots of a quadratic equation using various methods of functional testing.

Ans. (i) Boundary Value Analysis : Boundary condition means, an input value may be on the boundary, just below the boundary or just above the boundary.

Consider a program with two inputs variable x and y. There input variables have specified boundaries :

$$a \le x \le b$$
$$c \le y \le d$$

Hence both the input x and y are bounded by two intervals $[a, b]$ and $[c, d]$ respectively. For input x, we may design test cases with values a and b just above a and also just below b. Similarly for input y, we may have values c and d, just above c and also just below d.

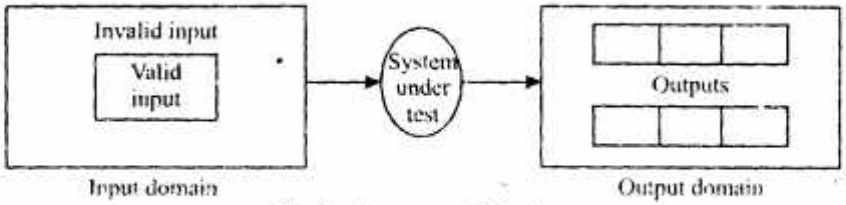


Input domain for program having two input variables

(ii) Equivalence Class Testing : In this method, input domain of a program is partitioned into a finite number of equivalence classes such that one can reasonably assume, but not be absolutely sure, that the test of a representative value of each class is equivalent to a test of any other value. Two steps are required in implementing this method :

(i) The equivalence classes are identified by taking each input condition and partitioning it into valid and invalid classes.

(ii) Generate the test cases using the equivalence classes identified in the previous step. This is performed by writing test cases covering all the valid equivalence classes. Then a test case is written for each invalid equivalence class so that no test contains more than one invalid class. This is to ensure that no two invalid classes mask each other.



Equivalence partitioning

Q. 7. (a) What is software reliability ? Explain any one reliability model in detail.

Ans. Software Reliability : Software reliability means operational reliability. It is also defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time.

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error. Hence it is the probability that the software will work without failure for a specified period of time in a given environment. Here environment and time is fixed. Reliability is for fixed time under given environment or stated conditions. So, reliability value is always for a well defined domain. Software reliability is the probability of a failure free operation of a program for a specified time in a specified environment. For example, a time-sharing system may have a reliability of 0.9s for 10 hr when employed by the average user. This system, when executed for 10hr, would operate without failure for 95 of there periods out of 100. As a result of the general way in which we defined failure, note that the concept of software reliability incorporates the notion of performance being satisfactory.

For example, excessive response time at a given load level may be considered unsatisfactory so that a routine must be recorded in more efficient form

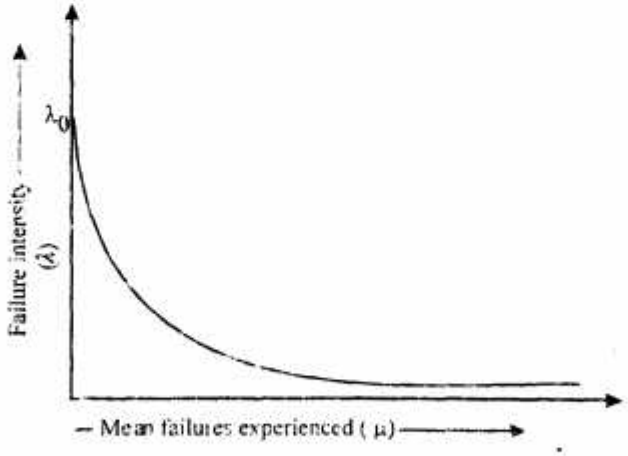
Logarithmic Poisson Execution Time Model :

This model is developed by Musa et. al. The failure intensity function is different here as compared to basic model. In this case, failure intensity function decreases exponentially whereas it is constant for basic model.

The failure intensity function is given as :

$$\lambda(\mu) = \lambda_0 \exp(-\theta_\mu)$$

where θ is called the failure intensity decay parameter. The relationship between failure intensity (λ) and mean failures experienced (μ) is shown in fig. (a)



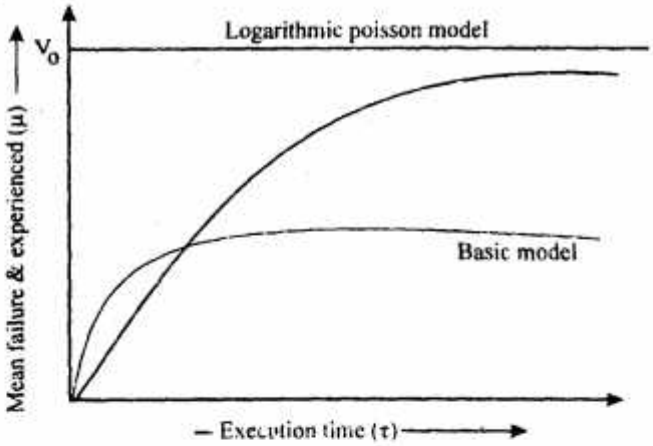
Relationship between μ and λ

The 'θ' represents the relative change of failure intensity per failure experienced. The slope of failure intensity function is :

$$\frac{d\lambda}{d\mu} = -\lambda_0 \theta \exp(-\mu\theta)$$

$$\frac{d\lambda}{d\mu} = -\theta\lambda$$

The relationship between execution time and mean failures experienced is given in fig. (b).



The expecied number of failures for this model is always infinite at infinite time. The relation for number of failures is given by

$$\mu(z) = \frac{!}{\theta} \ln(\lambda_0 \theta_z + 1)$$

The expression for failure intensity is given as :

$$\lambda(z) = \lambda_0 / (\lambda_0 \theta z + 1)$$

The relations for the additional number of failures and additional execution time in this model are :

$$\Delta u = \frac{1}{\theta} \ln\left(\frac{\lambda_p}{\lambda_F}\right)$$

&

$$\Delta z = \frac{1}{\theta}\left[\frac{1}{\lambda_F} - \frac{1}{\lambda_p}\right]$$

where, λ_p = Present failure intensity

λ_F = Failure intensity objective

Hence, at larger values of execution time, the logarithmic poisson model will have larger values of failure intensity than the basic model.

Q. 7. (b) Explain the various methods for software reviews.

Ans. Software Reviews : This is a popular requirements validation technique where a group of people will read the SRS document and look for possible problems :

(i) Plan Review : The review team is selected and time and place for review meeting is fixed.

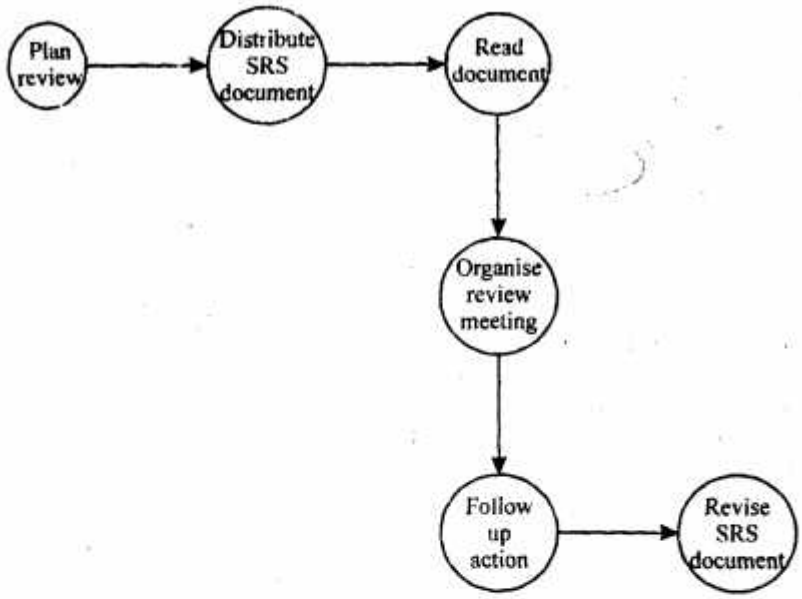
(ii) Distribute SRS Document : The SRS document is distributed to all the members.

(iii) Read SRS Document : Each member should read the document carefully to find conflict, omissions, inconsistencies, deviations from standards and other problems.

(iv) Organise Review Meeting : Each member presents his/her views and identified problems. The problems are discussed and a set of actions to address the problem is approved.

(v) Follow-up Actions : The chairperson of the team checks that the approved action have been carried out.

(vi) Revise SRS Document : The SRS document is revised to reflect the approved actions. At this stage, it may be accepted or may be reviewed.



Q. 8. Write short notes on the following :

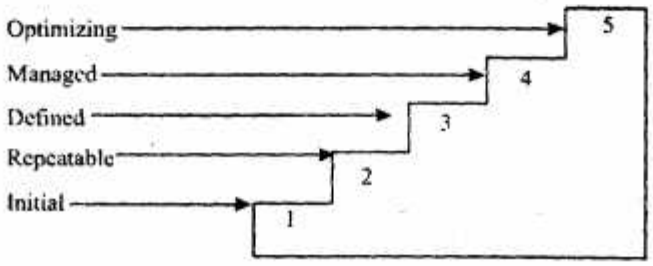
(a) CMM

(b) CASE Tools

(c) Data Dictionary

(d) Coupling and Cohesion

Ans. (a) CMM (Capability Maturity Model) : The Capability Maturity Model (CMM) is not a software life cycle model. Instead, it is a strategy for improving the software process, irrespective of the actual life cycle model used. The CMM was developed by Software Engineering Institute (SEI) of Carnegie-Mellon University, in 1986. CMM is used to judge the maturity of the software processes of an organization and to identify the key practices that are required to increase the maturity of these processes. The CMM is organized into five maturity levels as shown below :



Maturity Levels of CMM

(i) Initial (Maturity Level 1) : At this, the lowest level, there are essentially no sound software engineering management practices in place in the organization. Instead, everything is done on an adhoc basis.

(ii) Repeatable (Maturity Level 2) : At this level, policies for managing a software project and procedures to implement those policies are established. An objective in achieving level 2 is to institutionalize effective management processes for software projects, which allow organization to repeat successful practices developed on earlier projects, although the specific processes implemented by the projects may differ.

(iii) Defined (Maturity Level 3) : At this level, the standard process for developing and maintaining software across the organization is documented, including both software engineering and management processes. The software process capability of level 3 organizations can be summarized as "standard" and "consistent" because both software engineering and management activites are stable and repeatable.

(iv) Managed (Maturity Level 4) : At this level, the organization sets quantitative quality goals for both software products and processes. The software process capability at level 4 organizations can be summarized as "predictable" because the process is measured and operates within measurable limits.

(v) Optimizing (Maturity Level 5) : At this level, the entire organization is focused on continuous process improvement. The organizations have the means to identify weaknesses and strengthen the process proactively with the goal of preventing the occurrence of defects. The software process capability of level 5 organizations can be characterized as "continuously improving" because level 5 organizations are continuously striving to improve the range of their process capability, there by improving the process performance of their projects.

(b) CASE Tools : CASE stands for Computer Aided Software Engineering; it can be used to mean any computer-based tool for software planning, development and evolution.

(i) System Flowchart and ER-diagram Generation Tool :

What the Tool Does : Smartdraw is a perfect suite for drawing all kinds of diagrams and charts : Flowcharts, organizational charts, ER diagram etc. Tool tips automatically label buttons on the tool bar.

(ii) Data Flow Diagram Tool : The tool helps the user draw a standard data flow diagram for system analysis.

(iii) Tool to Convert Decision Table to Structured English : This table consists of a heading and four rows.

(iv) System Requirements Specification Documentation Tool : ARM or Automated Requirement Measurement tool aids in writing the system requirements specifications right.

(v) A Tool for Screen Design and Data Inputting : This tool is used to create the graphical user interface to describe the appearance and location of interface elements, you simply add prebuilt objects into place on screen.

(c) Data Dictionary : Data dictionary is simply repositories to store information about all data items defined in DFDS. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developer use the same definitions and terminologies. Typical information stored includes :

(i) Name of the data item

(ii) Aliases (other name for item)

(iii) Description/purpose

(iv) Related data items

(v) Range of values

(vi) Data structure definition/form

The name of the data item is self-explanatory. Aliases include other names by which this data item is called e.g., DEO for Data Entry Operator and DR for Deputy Registrar. Description/Purpose is a textual description of what the data item is used for or why it exists. Related data items capture relationships between data items e.g., total-marks must always equal to internal-marks plus external-marks.

Range of values records all possible values, e.g., total marks must be positive and between 0 to 100. Data flows capture the names of the processes that generate or receive the data item. If data item is primitive, then data structure definition/form captures the physical structure of the data item. If the data is itself a data aggregate, then data structure definition/form captures the composition of the data items in terms of other data items. The mathematical operators used within the data dictionary are defined in table given below :

Notation	Meaning
$x = a + b$	x consists of data elements a & b.
$x = [a / b]$	x consists of either data elements a or b.
$x = a$	x consists of an optional data element a.
$x = y\{a\}$	x consists of y or more occurrence of data element a.
$x = \{a\}z$	x consists of z or fewer occurrences of data element a.
$x = y\{a\}z$	x consists of some occurrences of data element a which are between y and z.

The da dictionary can be used to :

(i) Crete an ordered listing of all data items.

(ii) Cate an ordered listing of a subset of data items.

(iii) Fid a data item name from a description.

(iv) Esign tle software and test cases.

(d) Cuplingand Cohesion :

Couing : Cupling or dependency is the degree to which each program module relies on each one of the ther mdules.

Typs of Couling :

(i) Content Cupling : It is when one module modifies or relies on the internal working.

(ii) Common oupling : Common coupling is when two modules share the same global data.

(iii) External oupling : It occurs when two modules share an externally imposed data format or device interface.

(iv) Control Coupling : It is one module controlling the flow of another, by passing it information on what to do.

(v) Data Coupling : It is when modules share data through, for example, parameters.

Cohesion : Cohesion is a measure of how strongly-related the functionality expressed by the source code of a software module is

Types of Cohesion :

(i) Coincidental Cohesion : It is when parts of a module are grouped arbitrarily.

(ii) Logical Cohesion : It is when parts of a module are grouped because they logically are categorized to do the same thing, even if they are different by nature.

(iii) Procedural Cohesion : It is when parts of a module are grouped because hey always follow a certain sequence of execution.

(iv) Communication Cohesion : It is when parts of a module are grouped because they operate on the same data.

(v) Function Cohesion : It is when parts of a module are grouped because theyall contribute to a single well-defined task of the module.