SYLLABUS

Academic Session (2015-16)

SOFTWARE ENGINEERING [ETCS-303]

UNIT I

Introduction: Software Crisis, Software Processes, Software life cycle models: Waterfall, Prototype, Evolutionary and Spiral models, Overview of Quality Standards like ISO 9001, SEI-CMM.

Software Metrics: Size Metrics like LOC, Token Count, Function Count, Design Metrics, Data Structure Metrics, Information Flow Metrics.

[T1][R1][R2][No. of Hrs.: 10]

UNIT II

Software Project Planning: Cost estimation, static, Single and multivariate models, COCOMO model, Putnam Resource Allocation Model, Risk management.

Software Requirement Analysis and Specifications: Problem Analysis, Data Flow Diagrams, Data Dictionaries, Entity-Relationship diagrams, Software Requirement and Specifications, Behavioural and non-behavioural requirements, Software Prototyping.

[T1][R1][R2][No. of Hrs.: 11]

UNIT III

Software Design: Cohesion & Coupling, Classification of Cohesiveness & Coupling, Function Oriented Design, Object Oriented Design, User Interface Design.

Software Reliability: Failure and Faults, Reliability Models: Basic Model, Logarithmic Poisson Model, Calendar time Component, Reliability Allocation.

[T1][R1][R2] [No. of Hrs.: 12]

UNIT IV

Software Testing: Software process, Functional testing: Boundary value analysis, Equivalence class testing, Decision table testing,

Cause effect graphing, Structural testing: Path testing, Data flow and mutation testing, unit testing, integration and system testing, Debugging, Testing Tools & Standards.

Software Maintenance: Management of Maintenance, Maintenance Process, Maintenance Models, Reverse Engineering, Software Reengineering, Configuration Management, Documentation.

[T1][R1][R2] [No. of Hrs.: 11]

SYLLABUS

Code No.: ETCS-202	L	T	C
Paper: Software Engineering	3	1	4

Instructions to Paper Setters: **Maximum Marks: 75**

1. Question No. 1 should be compulsory and cover the entire syllabus. This question should have objective or short answer type questions. It should be of 25 marks.
2. Apart from question no. 1, rest of the paper shall consist of four units as per the syllabus. Every unit should have two questions. However, student may be asked to attempt only 1 question from each unit. Each question should be of 12.5 marks.

UNIT – I

Introduction: Software Crisis, Software Processes & Characteristics, Software life cycle models, Waterfall, Prototype, Evolutionary and Spiral Models

Software Requirements analysis & specifications: Requirement engineering, requirement elicitation techniques like FAST, QFD & Use case approach, requirements analysis using DFD, Data dictionaries & ER Diagrams, Requirements documentation, Nature of SRS, Characteristics & organization of SRS. **[No. of Hrs.: 11]**

UNIT – II

Software Project Planning: Size Estimation like lines of Code & Function Count, Cost Estimation Models, COCOMO, COCOMO-II, Putnam resource allocation model, Risk Management.

Software Design: Cohesion & Coupling, Classification of Cohesiveness & Coupling. Function Oriented Design, Object Oriented Design **[No. of Hrs.: 11]**

UNIT – III

Software Metrics: Software measurements: What & Why, Token Count, Halstead Software Science Measures, Design Metrics, Data Structure Metrics, Information Flow Metrics

Software Reliability: Importance, Hardware Reliability & Software Reliability, Failure and Faults, Reliability Models, Basic Model, Logarithmic Poisson Model, Software Quality Models, CMM & ISO 9001. **[No. of Hrs.: 11]**

UNIT – IV

Software Testing: Testing process, Design of test cases, functional testing: Boundary value analysis, Equivalence class testing, Decision table testing, Cause effect graphing, Structural testing, Path Testing, Data flow and mutation testing, Unit Testing, Integration and System Testing, Debugging, Alpha & Beta Testing, Testing Tools & Standards.

Software Maintenance: Management of Maintenance, Maintenance Process, Maintenance Models, Regression Testing, Reverse Engineering, Software Re-engineering, Configuration Management, Documentation. **[No. of Hrs.: 11]**

MODEL PAPER
FIFTH SEMESTER (B.TECH)
FIRST TERM EXAMINATION
SOFTWARE ENGINEERING [ETCS-303]

Time : 1 hrs. **M.M. : 30**

Note: *Attempt any three questions. Question No. 1 is compulsory.*

Q.1 (a) List the reasons for software crisis? Why case tools are not normally able to control it ?

Ans. (a) The major cause of the software crisis is that the machines have become several orders of magnitude more powerful. It linked to the overall complexity of hardware and the software development process. The crisis mainfested itself in several ways:

→ Projects running over–budget

→ Project running over-time

→ Software was very inefficient

→ Software was of low quality

→ Software often did not meet requirements.

→ Project were unmanageable and code difficult to maintain.

The case tools which are designed are for particular purpose. So, if we access the computer system for which CASE tools are available, you may be able to use them to design software. So if for the particular crisis a CASE tool is not designed yet then it will be difficult to control the crisis.

Q.1. (b) Statistically, the maximum percentage of errors belongs to which phase of SDLC.

Ans. The maximum percentage of errors belongs to specification phase of SDLC.

Q.1 (c) Define the terms Functional Requirements and Non-Functional Requirements.

Ans. Functional Requirements: Most requirements definition focuses mainly on functional requirements, which are based upon the expected functioning of the product or system to be created. Functioning, typically is equated with product/system features for which you might have a menu or button choice, such as: identify a customer, select an item to order, and calculate the amount due.

All things considered, requirements definers probably are best at identifying functional requirements, although they often overlook and get wrong more of the functional requirements than they ordinarily recognize. On hindsight reflection, they frequently do realize that many of the problems which surface later, and thus are harder and more expensive to fix, are attributable to inadequately addressed nonfunctional requirements.

Nonfunctional Requirements

Nonfunctional requirements refer to a whole slew of attributes including performance levels, security, and the various "ilities," such as usability, reliability, and availability. Invariably, requirements definers get wrapped up in how the product/system is expected to function and lose sight of these added elements.

When such factors are not addressed adequately, seemingly proper product/system functioning in fact fails to function. For example, a system may identify customers in such a slow, insecure, and difficult to use manner that it can cause mistakes which make data unreliable, provoke frustration-based attempted work-around that can create further problems, and ultimately lead to abandonment.

That's the recognized way in which nonfunctional requirements impact product/ system success. Other often unrecognized issues also need to be appreciated.

Q.1 (*d*) What is the difference between Function oriented and object oriented design.

Ans. 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: Functions are grouped together by which a higher level function is obtained, an eg:- of this technique isA/SD.

OOD: Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.

3. 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.

4. FOD approach is mainly used for computation sensitive application,

OOD: whereas OOD approach is mainly used for evolving system which mimics a business process or business case.

5. In FOD: we decompose in function/procedure level

OOD: we decompose in class level

6. FOD: Top down Approach

OOD: Bottom up approach

7. FOD: It views system as Black Box that performs high level function and later decompose it detailed function so to be mapped to modules.

OOD: Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.

8. FOD: Begins by considering the use case diagrams and Scenarios.

OOD: Begins by identifying objects and classes.

Q.1. (*e*) What are the advantages of using prototype. Also discuss various types of Prototyping.

Ans. The term prototype we mean small, useable working model of the system that in-corporates only same of the features that the user has given. Developing a prototype is an experimental procedure.

Prototypes are of two types:

(*i*) Thro away prototype (*ii*) Exploratory prototype.

Q.2 (*a*) What is software process? Why is it difficult to improve it?

Ans. The software process is a set of all activities that must be accomplished to build, deliver, deploy and evolve the software product. The software process broadly covers two types of activities (*a*) framework (*b*) umberella Activities.

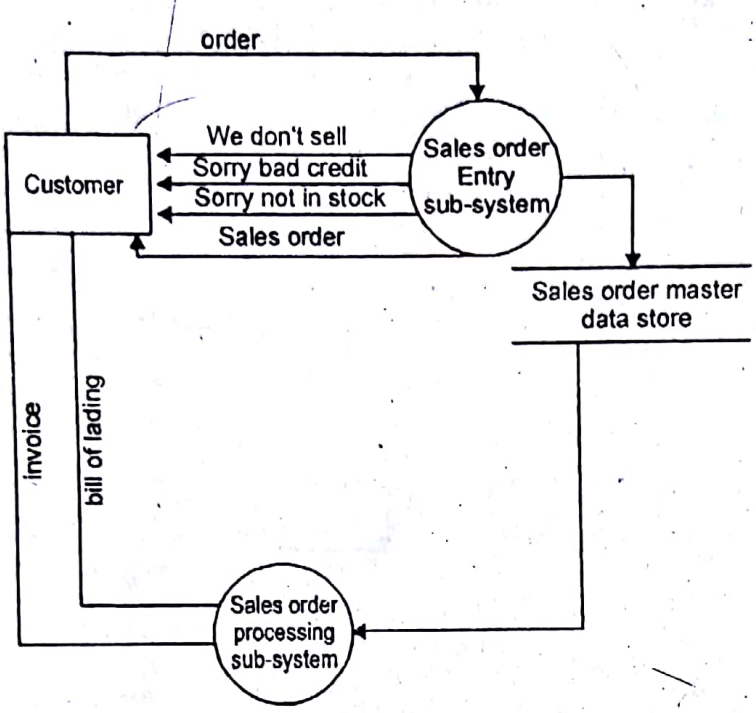
Frame work activities: This includes work tasks of software engineering, project milestones to show succesful completion of any task, final and intermediate airfacts and quality assurance points.

Umbrella activities: This includes software quality Assurance (SQA) software configuration management (SCM) and meaurement and metrics. It is difficult to improve it because the phases and activities in a development process are specified according to a model. These phases are the check point which indicate the completion of the intermediate tasks and not the final product. The entry to each phase specifies the

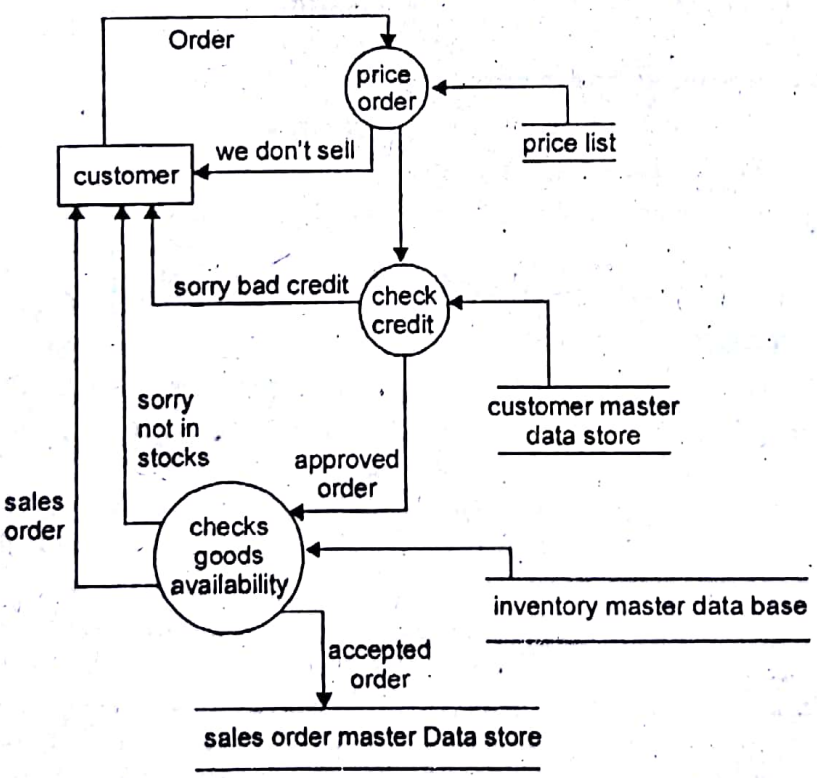
conditions of input to initiate that phase. The exit form each phase specifies some output criteria, which are the conditions that the intermediate work products should satisfy in order to terminate the activities of that phase. So it is difficult to improve this software process.

Q.2 (*b*) Draw level 0 and level 1 DFD of order processing system.

Ans.



Level 0 Logical Dataflow Diagram of the order processing system

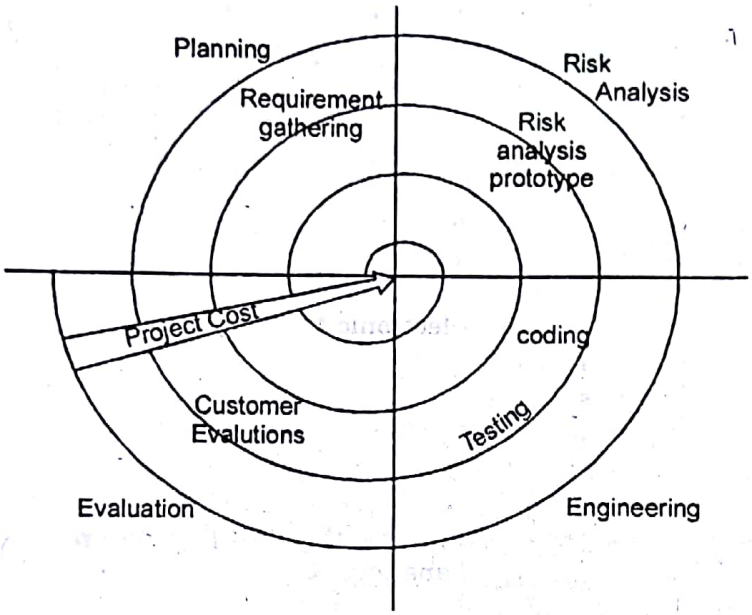


Level 1. DFD for order Processing system

Q.3 (*a*) Explain the spiral model of software Development. What are the limitations of such a model?

Ans 3 (*a*) Spiral model is a risk-driven process model generator for software projects. Based on unique risk patterns on a given project, the spiral model guides a term to adopt elements of one or more process models, such incremental, waterfall all or evolutionary prototyping. A softwre project repeatedly passes through these phase in interations. The baseline spiral, starting in the planning phase, requirements are gathered and risk is accessed. Each subsequent spirals builds on the baseline spiral.

Phases of spiral model : Planning phase, Risk Analysis, Engineering phase, Evaluation phase.



limitations of spiral model:-

When costs and risk evaluation is important

For medium to high-risks projects.

Long-term project commitment unwise because of potential changes to economic priorities.

Users are unsure of their needs.

Requirements are complex.

New product line, significant changes are expected.

Q.3 (*b*) What is requirement Engineering process? Discuss FAST and QFD techniques of requirement elicitation.

Ans. Requirements engineering (RE) refers to the process of defining, documenting and maintaining requirements and to the subfields of systems engineering and software engineering concerned with this process.

Facilitated Application Specification Techniques Too often, customers and software engineers have an unconscious "us and them" mind-set. Rather than working as a team to identify and refine requirements, each constituency defines its own "territory" and communicates through a series of memos, formal position papers, documents, and question and answer sessions. History has shown that this approach doesn't work very well.

Misunderstandings abound, important information is omitted, and a successful working relationship is never established. It is with these problems in mind that a number of independent investigators have developed a team-oriented approach to requirements gathering that is applied during early stages of analysis and specification. Called facilitated application specification techniques "(FAST).

This approach encourages the creation of a joint team of customers and developers who work together to:

Identify the problem

Propose elements of the solution Negotiate different approaches and specify a preliminary set of solution requirements.

FAST has been used predominantly by the information systems community, but the technique offers potential for improved communication in applications of all kinds. Many different approaches to FAST have been proposed. Each makes use of a slightly different scenario, but all apply some variation on the following basic guidelines:

- A meeting is conducted at a neutral site and attended by both software engineers and customers. - Rules for preparation and participation are established. - An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas. - A 'facilitator' (can be a; customer, a developer, or an outsider) controls the meeting. - A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used. The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal. To better understand the flow of events as they occur in a typical FAST meeting, we present a brief scenario that outlines the sequence of events that lead up to the meeting, occur during the meeting, and follow the meeting. Initial meetings between the developer and customer occur and basic questions and answers help to establish the scope of the problem and the overall perception of a solution. Out of these initial meetings, the developer and customer write a one- or two- page "product request." A meeting place, time, and date for FAST are selected and a facilitator is chosen. Attendees from both the development and customer/user organizations are invited to attend.

The product request is distributed to all attendees before the meeting date.

3. Quality Function Deployment

A quality management technique that translates needs of customers into technical requirements of software.

Normal Requirement: meeting objectives & goals stated for a product or system during meeting

Expected Requirement: Implicit to products / system and may be so fundamental that customer does not explicitly state them

Exciting Requirement: Features beyond customer's expectation and prove to be very satisfying when present

Q.4 (*a*) Define module cohesion. Explain different types of cohesion.

Ans. Module cohesion refors to the degree to which the elements of a module belong together. It is measure of two strongly related each piece of functionality expressed by the source code of a software module is.

Cohesion is a qualitative measure, meaning that the sourco code to be meaured is examined using a rubric to determine a classification. Cohesion types are as follows:

Coincidental Cohesion: Coincidental cohesion is when parts of module are grouped arbitarily, the only relationhip between the parts is that they have been grouped together.

logical cohesion:- logical cohesion is when parts of a module are grouped because they are logically categorized to do the same thing, even if they are different by nature.

Temporal Cohesion:- Temporal cohesion is when parts of a module are grouped by when they are processed, the parts are processed at a particular time in program execuction.

Procedural cohesion:- Procedureal cohesion is when parts of a module are grouped because they always follow a certain sequence of execution.

Communictional informational cohesion:- Communicational cohesion is when parts of module are grouped because they operate on the same date

Sequential cohesion:- Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line

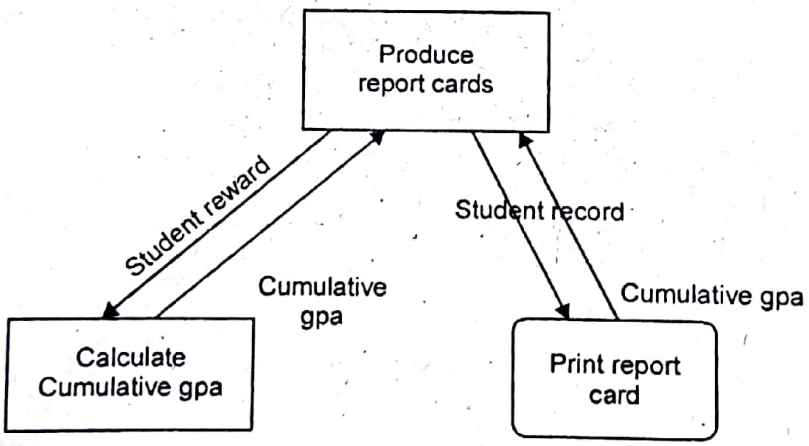
Functional Cohesion:- Functional cohesion is when parts of a module are grouped because they all contribute to a single well defined task of a module.

Q.4 (*b*) Explain stamp coupling with Example.

Ans. Coupling is the manner and degree of inter dependence between software modules, a measure of how closely connected two modules are the strength of the relationships between modules.

Stamp coupling occurs when modules share a composite data structure and use only a part of it, possibly a different part. This may lead to changing the way a module reads a record because a field that the module does not need has been modified.

Example



Here we assume the "student record" contains name, address, S.S.N, outside activities, medical information, contact names etc..... in addition to academic performance information.

Q.4 (*c*) List the charactersties of good SRS.

Ans. Characteristics of good SRS.

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

2. Consistent: System functions and performance level must be compatible and the required quality features must not contradict the utility of the system. For example,

the only aircraft that is totally safe is one that cannot be started, contains no fuel or other liquid, and is securely lied down.

3. Correct: The specifications must define the desire of capabilities real world operational enviroment, its interface to that enviroment and its iteration with that enviroment. It is the real world aspect of requirements that is the major source of difficulty in acheving specification correctness.

4. Modifiable : Related concerns must be grouped together and unrelated concerns must be separated. Requirement document must have a logical structure to be modifiable.

5. Ranked: Ranking specification statements according to stability and importance is established in the requirements documents's organization and structure

6. Testable : A requirement's specification must be stated in such a manner that one can test it against pass/fail or quantitative itself or referenced information.

7. Traceable : Each requirement stated within the SRS document must be uniquely identified to achieved Traceability.

8. Unambiguous: A statements of a requirements is unambiguous it can only be interpreted one way.

9. Valid: To validate a requirements specification all the project participants, managers, engineers and customer representation must be able to understand.

10. Verifiable: In order to be verifiable, requirement specifications at one level of abstraction must be consistent with those at another level of abstraction.

MODEL PAPER
FIFTH SEMESTER (B.TECH)
SECOND TERM EXAMINATION
SOFTWARE ENGINEERING [ETCS-303]

Time : 1 hrs. **M.M. : 30**

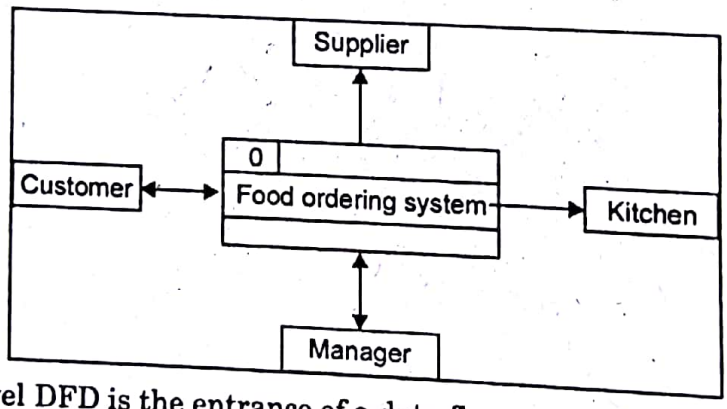
Note: *Attempt any three questions. Question No.1 is compulsory.*

Q.1 (a) Explain DFD with Example. **(5)**

Ans. Data Flow Diagram (DFD) provides a visual representation of the flow of information (i.e. data) within a system. By drawing a Data Flow Diagram, you can tell the information provided by and delivered to someone who takes part in system processes, the information needed in order to complete the processes and the information needed to be stored and accessed.

The Food Ordering System (Example)

(Level 0 or) Context Level DFD: The figure below shows a context Data Flow Diagram that is drawn for a Food Ordering System. It contains a process (shape) that represents the system to model, in this case, the "*Food Ordering System*". It also shows the participants who will interact with the system, called the external entities. In this example, *Supplier*, *Kitchen*, *Manager* and *Customer* are the entities who will interact with the system. In between the process and the external entities, there are data flow (connectors) that indicate the existence of information exchange between the entities and the system.



Context Level DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

Q.1 (b) Explain Putnam Resource allocation model. **(5)**

Ans. Putnam model describes the *time* and *effort* required to finish a software project of specified *size*. SLIM (Software LIfecycle Management) is the name given by Putnam to the proprietary suite of tools his company QSM, Inc. has developed based on his model. It is one of the earliest of these types of models developed, and is among the most widely used. Closely related software parametric models is Constructive Cost Model (COCOMO).

Putnam used his observations about productivity levels to derive the software equation:

$$\frac{B^{1/3}.\text{Size}}{\text{Productivity}} = \text{Effort}^{1/3}.\text{Time}^{4/3}$$

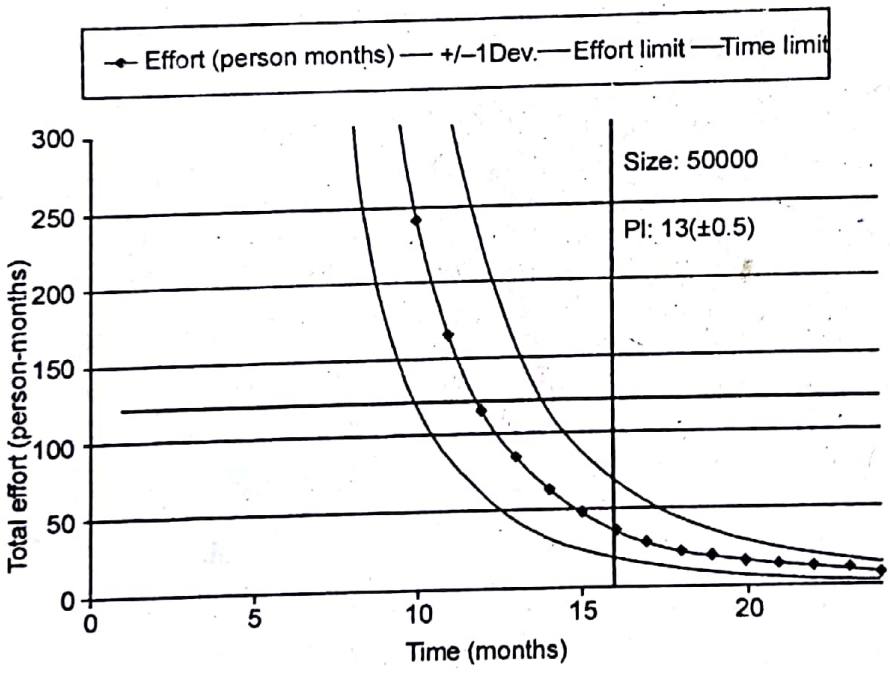
where:

- Size is the product size (whatever size estimate is used by your organization is appropriate). Putnam uses ESLOC (Effective Source Lines of Code) throughout his books.
- B is a scaling factor and is a function of the project size.
- Productivity is the **Process Productivity**, the ability of a particular software organization to produce software of a given size at a particular defect rate.
- Effort is the total effort applied to the project in person-years.
- Time is the total schedule of the project in years.

In practical use, when making an estimate for a software task the software equation is solved for *effort*:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity}.\ \text{Time}^{4/3}}\right]^3 .B$$

An estimated software size at project completion and organizational process productivity is used. Plotting *effort* as a function of *time* yields the *Time-Effort Curve*. The points along the curve represent the estimated total effort to complete the project at some *time*. One of the distinguishing features of the Putnam model is that total effort decreases as the time to complete the project is extended. This is normally represented in other parametric models with a schedule relaxation parameter.



This estimating method is fairly sensitive to uncertainty in both *size* and *process productivity*. Putnam advocates obtaining process productivity by calibration:

$$\text{Process Productivity} = \frac{\text{Size}}{\left[\frac{\text{Effort}}{B}\right]^{1/3}.\text{Time}^{4/3}}$$

Putnam makes a sharp distinction between 'conventional productivity' : *size / effort* and process productivity.

One of the key advantages to this model is the simplicity with which it is calibrated. Most software organizations, regardless of maturity level can easily collect *size*, *effort* and duration (*time*) for past projects. Process Productivity, being exponential in nature is typically converted to a linear *productivity index* an organization can use to track their own changes in productivity and apply in future effort estimates.

Q.2 (a) Write a short note on Risk management.

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. Risk management's objective is to assure uncertainty does not deflect the endeavor from the business goals.

Risks can come from various sources: e.g., uncertainty in financial markets, threats from project failures (at any phase in design, development, production, or sustainment life-cycles), legal liabilities, credit risk, accidents, natural causes and disasters as well as deliberate attack from an adversary, or events of uncertain or unpredictable root-cause. There are two types of events i.e. negative events can be classified as risks while positive events are classified as opportunities. Several risk management standards have been developed including the Project Management Institute, the National Institute of Standards and Technology, actuarial societies, and ISO standards. Methods, definitions and goals vary widely according to whether the risk management method is in the context of project management, security, engineering, industrial processes, financial portfolios, actuarial assessments, or public health and safety.

Risk sources are more often identified and located not only in infrastructural or technological assets and tangible variables, but also in human factor variables, mental states and decision making. The interaction between human factors and tangible aspects of risk highlights the need to focus closely on human factors as one of the main drivers for risk management, a "change driver" that comes first of all from the need to know how humans perform in challenging environments and in face of risks. As the author describes, «it is an extremely hard task to be able to apply an objective and systematic self-observation, and to make a clear and decisive step from the level of the mere "sensation" that something is going wrong, to the clear understanding of how, when and where to act. The truth of a problem or risk is often obfuscated by wrong or incomplete analyses, fake targets, perceptual illusions, unclear focusing, altered mental states, and lack of good communication and confrontation of risk management solutions with reliable partners. This makes the Human Factor aspect of Risk Management sometimes heavier than its tangible and technological counterpart.

Strategies to manage threats typically include transferring the threat to another party, avoiding the threat, reducing the negative effect or probability of the threat, or even accepting some or all of the potential or actual consequences of a particular threat, and the opposites for opportunities.

Q.2 (*b*) Draw ER Diagram for ATM management system.

Ans. The General Things needed in a Banking System are: -

1. Person Opens an Account
2. Person using ATM for Transaction

The person opens an Account in a Bank and gets a account number and ATM card. The person can make transactions in ATM centres. The Details of the Transaction has to be maintained Between Three Entity. i.e. User, Account, ATM

User Table:

User_ID	Name (First_Name, Last_Name	Address	Contact_Number

Account Table:

Account_Number	User_ID	Account_Type (Saving_Account, Current_Account, Over_draft_Account)	Balance

ATM Table

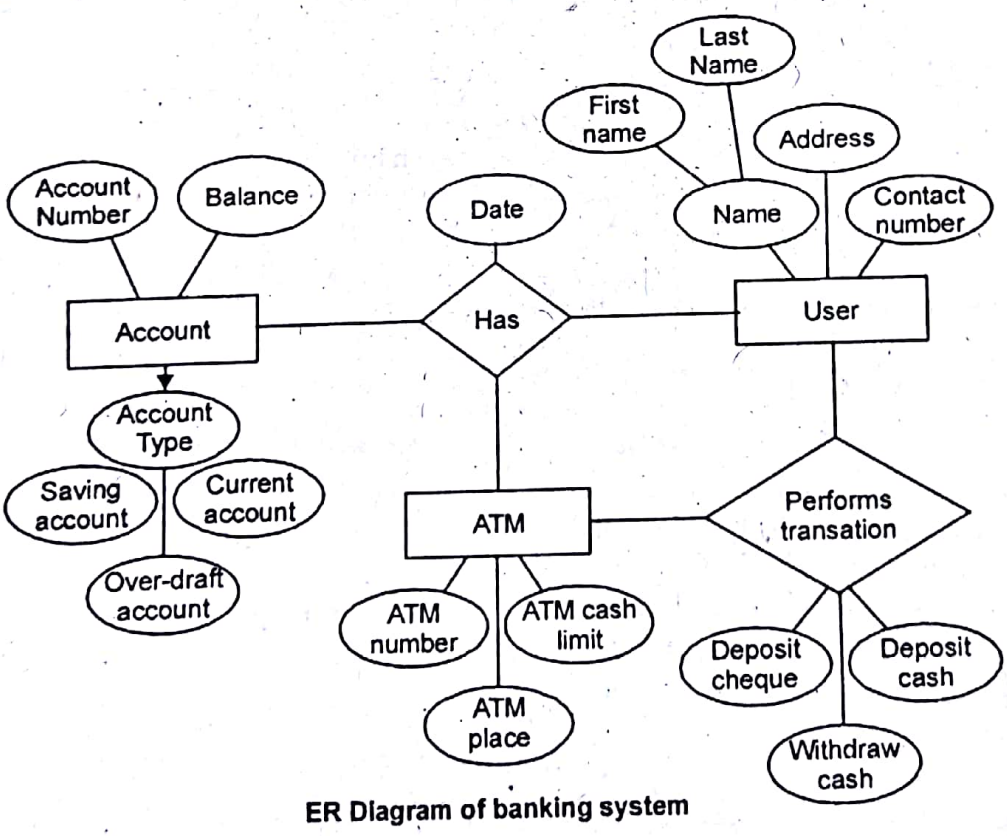
ATM_Number	ATM_Place	ATM_Cash_Limit

Opening_Account Table:

Date	User_ID	Account_Number	ATM_Number

Transaction Table

Date	User_ID	Account_Number	ATM_Number	Transaction_Type (Deposit _Cheque,Deposit_Cash, Withdraw_Cash)



ER Diagram of banking system

Q.3 (a) What is software Prototyping. Explain?

Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an

activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering or manufacturing.

A prototype typically simulates only a few aspects of, and may be completely different from, the final product.

Prototyping has several benefits: The software designer and implementer can get valuable feedback from the users early in the project. The client and the contractor can compare if the software made matches the software specification, according to which the software program is built. It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met.

Throwaway prototyping

Also called close-ended prototyping. Throwaway or Rapid Prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system.

Evolutionary prototyping

Evolutionary Prototyping (also known as breadboard prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built.

When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt.

"..evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood."

Q.3. (b) Explain Reliability allocation.

Ans. Reliability Allocation is a method of apportioning a system target reliability amongst sub-systems and components. It is a useful tool at design stage for determining the required reliability of equipment to achieve a given system reliability target.

Using Reliability Allocation: During the design phase of a product, it is often required to evaluate the reliability of the system. The question of how to meet a reliability goal for the system arises. Reliability Allocation can be used to set reliability goals for individual subsystems so that this goal is met. The simplest method is simply to distribute the reliability goal equally among all subsystems.

However, this rarely gives the best distribution of reliability objectives for all subsystems. It is better to allocate reliability values between the subsystems based on complexity, criticality, achievable reliability, or other factors that are deemed appropriate.

The Allocation module within Reliability Workbench supplies the user with six methods for assigning subsystem reliability values:

- Non-restricted equal allocation
- Non-restricted graded allocation
- Non-restricted proportional allocation
- Non-restricted redundancy proportional allocation
- Non-restricted reliability re-allocation
- Restricted direct research allocation

Inside the module, a system hierarchy can be constructed where sub-systems may be broken down further to component level, allowing a complete system allocation model.

The Reliability Growth module of Reliability Workbench analyzes test data by calculating scale and shape parameters that define a growth curve that fits the data. The scale and shape parameters can be used to calculate failure intensity, Mean Time to Failure (MTTF), or Unreliability at an arbitrary time. This allows a user to determine there is an improving or worsening trend in reliability in a system.

The Reliability Growth module can match a curve to continuous or discrete test times, based on one of the following methods:

- Power Law
- Power Law (grouped failures)
- Crow Discrete (Power Law for discrete data)

Reliability Workbench automatically fits a curve to the data according to the chosen method and displays the results graphically in the form of cumulative number of failures plots, failure intensity plots, MTTF plots, and Unreliability plots.

Data may be entered manually by the user or imported from other packages or transferred via the Windows clipboard.

New data can be analyzed in 3 simple steps:

- Enter or import the data
- Choose the appropriate calculation options for the data
- Assign the appropriate Growth method

Q.4 (a) Explain Logarithmic Poisson Model in detail.

Ans. Poisson regression is a form of regression analysis used to model count data and contingency tables. Poisson regression assumes the response variable Y has a Poisson distribution, and assumes the logarithm of its expected value can be modeled by a linear combination of unknown parameters. A Poisson regression model is sometimes known as a log-linear model, especially when used to model contingency tables.

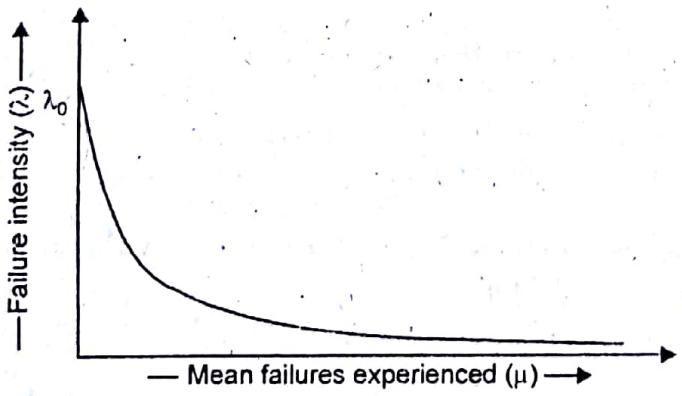
Poisson regression models are generalized linear models with the logarithm as the (canonical) link function, and the Poisson distribution function as the assumed probability distribution of the response.

The logarithmic Poisson model may be viewed as a continuous version of the geometric model.

- **Logarithmic Poison Execution Time Model**

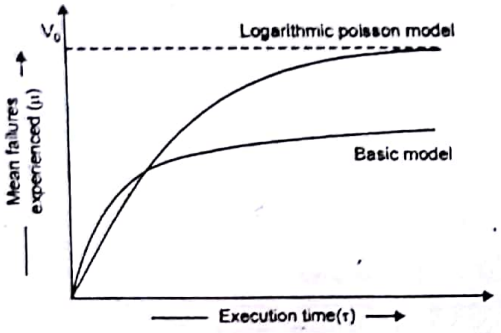
Failure Intensity

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



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

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



$$\mu(\tau) = \frac{1}{\theta}Ln(\lambda_0\theta\tau + 1)$$

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

$$\Delta\mu = \frac{1}{\theta}Ln\left(\frac{\lambda_p}{\lambda_F}\right)$$

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

λ_F = Present failure intensity

λ_p = Failure intensity objective

Q. 4. (*b*) Explain Cohesiveness and Coupling.

Ans. Coupling: An indication of the strength of interconnections between program units.

Highly coupled have program units dependent on each other. Loosely coupled are made up of units that are independent or almost independent.

Modules are independent if they can function completely without the presence of the other. Obviously, can't have modules completely independent of each other. Must interact so that can produce desired outputs. The more connections between modules, the more dependent they are in the sense that more info about one modules is required to understand the other module.

Three factors: number of interfaces, complexity of interfaces, type of info flow along interfaces.

Want to minimize number of interfaces between modules, minimize the complexity of each interface, and control the type of info flow. An interface of a module is used to pass information to and from other modules.

In general, modules tightly coupled if they use shared variables or if they exchange control info.

Loose coupling if info held within a unit and interface with other units via parameter lists. Tight coupling if shared global data.

If need only one field of a record, don't pass entire record. Keep interface as simple and small as possible.

Two types of info flow: data or control.

• Passing or receiving back control info means that the action of the module will depend on this control info, which makes it difficult to understand the module.

• Interfaces with only data communication result in lowest degree of coupling, followed by interfaces that only transfer control data. Highest if data is hybrid.

Ranked highest to lowest:

1. Content coupling: if one directly references the contents of the other.

When one module modifies local data values or instructions in another module. (can happen in assembly language)

if one refers to local data in another module.

if one branches into a local label of another.

2. Common coupling: access to global data.

modules bound together by global data structures.

3. Control coupling: passing control flags (as parameters or globals) so that one module controls the sequence of processing steps in another module.

4. Stamp coupling: similar to common coupling except that global variables are shared selectively among routines that require the data. More desirable than common coupling because fewer modules will have to be modified if a shared data structure is modified. Pass entire data structure but need only parts of it.

5. Data coupling: use of parameter lists to pass data items between routines.

Cohesion: Measure of how well module fits together.

A component should implement a single logical function or single logical entity. All the parts should contribute to the implementation.

Many levels of cohesion:

1. Coincidental cohesion: The parts of a component are not related but simply bundled into a single component. Harder to understand and not reusable.

2. Logical association: similar functions such as input, error handling, etc. put together. Functions fall in same logical class. May pass a flag to determine which ones executed. Interface difficult to understand. Code for more than one function may be intertwined, leading to severe maintenance problems. Difficult to reuse

3. Temporal cohesion: all of statements activated at a single time, such as start up or shut down, are brought together. Initialization, clean up.

Functions weakly related to one another, but more strongly related to functions in other modules so may need to change lots of modules when do maintenance.

4. Procedural cohesion: a single control sequence, e.g., a loop or sequence of decision statements. Often cuts across functional lines. May contain only part of a complete function or parts of several functions.

Functions still weakly connected, and again unlikely to be reusable in another product.

5. Communicational cohesion: operate on same input data or produce same output data. May be performing more than one function. Generally acceptable if alternate structures with higher cohesion cannot be easily identified.

still problems with reusability.

6. Sequential cohesion: output from one part serves as input for another part. May contain several functions or parts of different functions.

7.Informational cohesion: performs a number of functions, each with its own entry point, with independent code for each function, all performed on same data structure. Different than logical cohesion because functions not intertwined.

8. Functional cohesion: each part necessary for execution of a single function. e.g., compute square root or sort the array.

9. Type cohesion: modules that support a data abstraction.

MODEL PAPER
FIFTH SEMESTER (B.TECH)
END TERM EXAMINATION
SOFTWARE ENGINEERING [ETCS-303]

Time : 3 hrs. **M.M. : 75**

Note: *Question One is compulsory attempt any four from the rest.*

Q.1. (*a*) The most important feature of spiral model is **(1.5 marks)**

Ans: Risk management.

Q.1. (*b*) The worst type of coupling is

Ans: Content coupling

Q.1. (*c*) One of the fault base testing techniques is

Ans: Mutation testing

Q.1(*d*) Changes made to an information system to add the desired but not necessarily the required features is called

Ans: Perfective maintenance.

Q.1.(*e*) All the modules of the system are integrated and tested as complete system in the case of

Ans: Big-Bang testing

Q.1(*f*) If every requirement stated in the Software Requirement Specification (SRS) has only one interpretation, SRS is said to be

Ans: Unambiguous.

Q.1(*g*) A fault simulation testing technique is

Ans: Mutation testing

Q.1 (*h*) Modules X and Y operate on the same input and output data, then the cohesion is

Ans: Communicational

Q.1 (*i*) Explain CMM and its levels. (10)

Ans. A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

In CMMI models with a staged representation, there are five maturity levels designated by the numbers 1 through 5

1. Initial
2. Managed
3. Defined
4. Quantitatively Managed
5. Optimizing

Maturity Level 1-Initial

At maturity level 1, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes.

Maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.

Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

Maturity Level 2-Managed

At maturity level 2, an organization has achieved all the **specific** and **generic goals** of the maturity level 2 process areas. In other words, the projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.

The process discipline reflected by maturity level 2 helps to ensure that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans.

At maturity level 2, requirements, processes, work products, and services are managed. The status of the work products and the delivery of services are visible to management at defined points.

Commitments are established among relevant stakeholders and are revised as needed. Work products are reviewed with stakeholders and are controlled.

The work products and services satisfy their specified requirements, standards, and objectives.

Maturity Level 3-Defined

At maturity level 3, an organization has achieved all the **specific** and **generic goals** of the process areas assigned to maturity levels 2 and 3.

At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures. At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project). At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit. The organization's set of standard processes includes the processes addressed at maturity level 2 and maturity level 3. As a result, the processes that are performed across the organization are consistent except for the differences allowed by the tailoring guidelines.

Another critical distinction is that at maturity level 3, processes are typically described in more detail and more rigorously than at maturity level 2. At maturity level 3, processes are managed more proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.

Maturity Level 4-Quantitatively Managed

At maturity level 4, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, and 4 and the **generic goals** assigned to maturity levels 2 and 3.

At maturity level 4 Subprocesses are selected that significantly contribute to overall process performance. These selected subprocesses are controlled using statistical and other quantitative techniques.

Quantitative objectives for quality and process performance are established and used as criteria in managing processes. Quantitative objectives are based on the needs

of the customer, end users, organization, and process implementers. Quality and process performance are understood in statistical terms and are managed throughout the life of the processes.

For these processes, detailed measures of process performance are collected and statistically analyzed. Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences.

Quality and process performance measures are incorporated into the organization.s measurement repository to support fact-based decision making in the future.

A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

Maturity Level 5-Optimizing

At maturity level 5, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, 4, and 5 and the **generic goals** assigned to maturity levels 2 and 3.

Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.

Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements.

Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement.

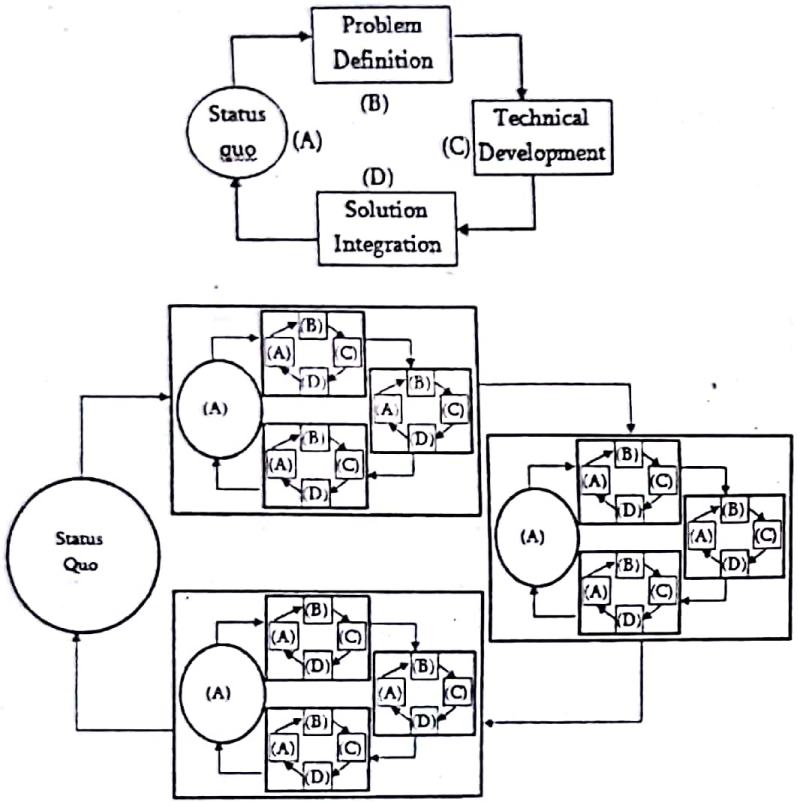
The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives. Both the defined processes and the organization's set of standard processes are targets of measurable improvement activities.

Optimizing processes that are agile and innovative depends on the participation of an empowered workforce aligned with the business values and objectives of the organization. The organization's ability to rapidly respond to changes and opportunities is enhanced by finding ways to accelerate and share learning. Improvement of the processes is inherently part of everybody's role, resulting in a cycle of continual improvement.

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 special causes of process variation and providing statistical predictability of the results. 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 common causes of process variation and changing the process (that is, shifting the mean of the process performance) to improve process performance (while maintaining statistical predictability) to achieve the established quantitative process-improvement objectives.

Q.1.(j) What are the phases of problem solving loop? **(3)**

Ans. All software development can be characterized as a problem solving loop in which four distinct stages are encountered: status quo, problem definition, technical development, and solution integration.



Status quo represents the current state of affairs, problem definition identifies the specific problem to be solved, technical development solves the problem through the application of some technology and solution integration delivers the results such as documents, programs, data, new business function, new product etc to those who requested the solution in the first place. The generic software engineering phases and steps easily map into these stages.

This problem solving loop applies to software engineering work at many different levels of resolution. It can be used at the macro level when the entire application is considered, at a mid-level when program components are being engineered, and even at the line of code level. Therefore, a fractal representation can be used to provide an idealized view of process. In Figure each stage in the problem solving loop contains an identical problem solving loop, which contains still another problem solving loop and this continues to some rational boundary; for software, a line of code.

The simplified view of figure leads to a very important idea: regardless of the process model that is chosen for a software project, all of the stages—status quo, problem definition, technical development, and solution integration—coexist simultaneously at some level of detail.

Q.2 Explain in detail COCOMO, with cost drivers. **(12.5)**

Ans. 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

Basic COCOMO compute 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) = $a_b(KLOC)^{b_b}$ **[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 COCOMOs

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

Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value). An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an *effort adjustment factor (EAF)*. Typical values for EAF range from 0.9 to 1.4.

Cost Drivers	Ratings					
	Very Low	Low	Nominal	Very High	Extra High	High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

The Intermediate Cocomo formula now takes the form:

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

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 a_i and the exponent b_i are given in the next table.

Software project	a_i	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.

Q.3. (a) Explain RAD model. **(7)**

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.

Business modeling: The information flow is identified between various business functions.

Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.

Process modeling: Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.

Application generation: Automated tools are used to convert process models into code and the actual system.

Testing and turnover: Test new components and all the interfaces.

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 code generation is very high.
- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Q.3 (b) Define: error, bug, fault and failure and defect. **(5.5)**

Ans. Error: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. This can be a misunderstanding of the internal state of the software, an oversight in terms of memory management, confusion about the proper way to calculate a value, etc.

Failure: The inability of a system or component to perform its required functions within specified performance requirements. See: bug, crash, exception, and fault.

Bug: A fault in a program which causes the program to perform in an unintended or unanticipated manner. See: anomaly, defect, error, exception, and fault. Bug is terminology of Tester.

Fault: An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. See: bug, defect, error, exception.

Defect: Commonly refers to several troubles with the software products, with its external behavior or with its internal features.

Q.4 (a) Difference between verification and validation also explain mutation testing. **(6)**

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.

Objective	To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.	To ensure that the product actually meets the user's need, and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment.
Question	Are we building the product right?	Are we building the right product?
Evaluation Items	Plans, Requirement Specs, Code, Test Cases	The actual product/software.
Activities	• Reviews • Walkthroughs • Inspections	• Testing

Mutation testing is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in small ways. Each mutated version is called a *mutant* and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. This is called *killing* the mutant. Test suites are measured by the percentage of mutants that they kill. New tests can be designed to kill additional mutants. Mutants are based on well-defined *mutation operators* that either mimic typical programming errors (such as using the wrong operator or variable name) or force the creation of valuable tests (such as dividing each expression by zero). The purpose is to help the tester develop effective tests or locate weaknesses in the test data used for the program or in sections of the code that are seldom or never accessed during execution.

Q.4. (b) Explain characteristics of a good SRS. **(6.5)**

Ans. Software requirement specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it. The basic goal of the requirement phase is to produce the SRS, Which describes the complete behavior of the proposed software. SRS is also helping the clients to understand their own needs.

Advantages:

Software SRS establishes the basic for agreement between the client and the supplier on what the software product will do.

1. A SRS provides a reference for validation of the final product.
2. A high-quality SRS is a prerequisite to high-quality software.
3. A high-quality SRS reduces the development cost.

Characteristics of an SRS

1. Correct
2. Complete
3. Unambiguous
4. Verifiable
5. Consistent
6. Ranked for importance and/or stability
7. Modifiable
8. Traceable

An SRS is correct if every requirement included in the SRS represents something required in the final system. An SRS is complete, if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS. Correctness ensures that what is specified is done correctly, completeness ensures that everything is indeed specified.

An SRS is unambiguous if and only if every requirement stated has one and only one interpretation. Requirements are often written in natural language, which are inherently ambiguous.

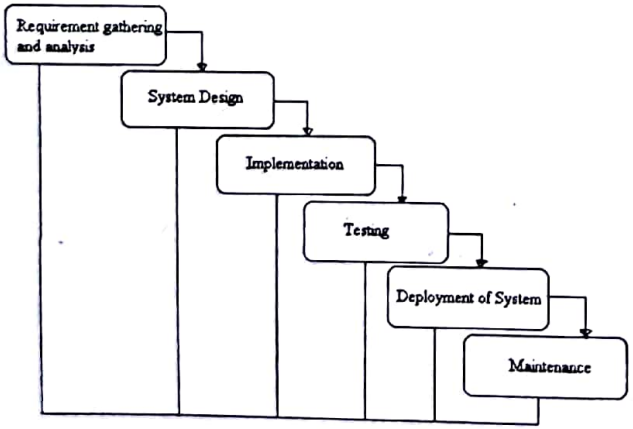
An SRS is verifiable if and only if every stated requirement is verifiable. A requirement is verifiable if there exists some cost-effective process that can check whether the final software meets that requirement. An SRS is consistent if there is no requirement that conflicts with another.

Terminology can cause inconsistencies; for example, different requirements may use different terms to refer to the same object. All the requirements for software are not of equal importance. Some are critical, others are important but not critical, and there are some, which are desirable, but not very important. An SRS is ranked for importance and the stability of the requirement are indicated. Stability of requirement reflects the chances of it changing in future. An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development. Forward traceability means that each requirement should be traceable to some design and code elements. Backward traceability requires that it be possible to trace design and code elements to the requirements they support. Traceability aids verification and validation.

Q.5. (α) Explain waterfall model and also Explain testing activity throughout the process of V- Model. **(5.5)**

Ans. The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

General Overview of "Waterfall Model"

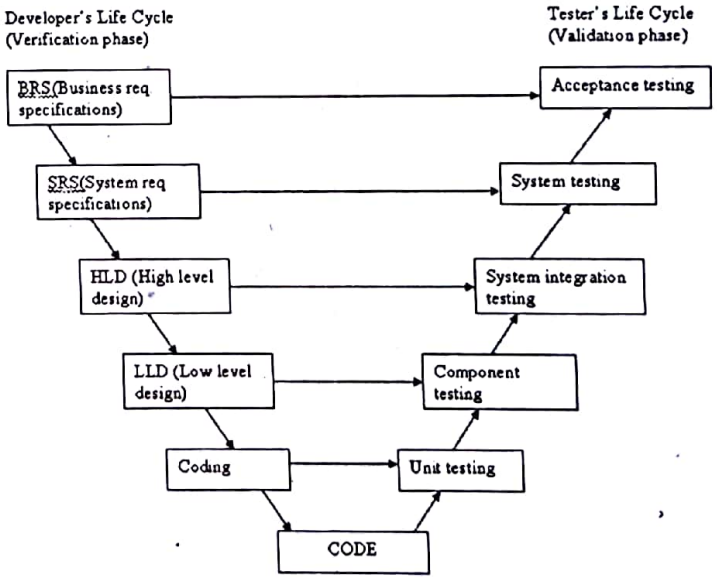


Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development.



- The various phases of the V-model are as follows:
- **Requirements** like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a system test plan is created.

The test plan focuses on meeting the functionality specified in the requirements gathering.

- **The high-level design (HLD)** phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.
- **The low-level design (LLD)** phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. Component tests are created in this phase as well.
- **The implementation** phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.
- **Coding:** This is at the bottom of the V-Shape model. Module design is converted into code by developers.

Q.5 (b) Explain NR Curve. **(7)**

Ans. The **Putnam–Norden–Rayleigh curve**, also known as the **PNR curve**, is an equation specifying the relationship between applied effort and delivery time for a software project. A PNR curve can be used to determine the least cost time for delivery t_0 up to the limit t_{min} the absolute minimal amount of time required to complete the project no matter how many human resources are added. The PNR curve was used to derive the Software equation.

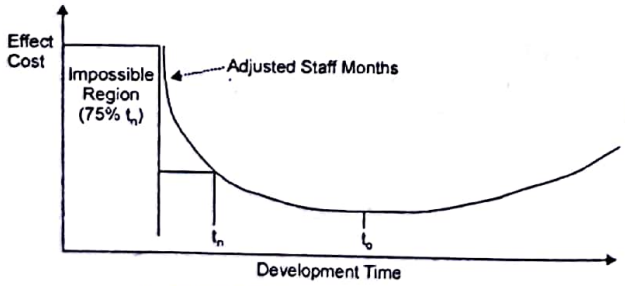
The following equation shows the relationship of project effort as a function of project delivery time.

$$E_a = m\left(\frac{t_d^4}{t_a^4}\right)$$

Where

- E_a = Effort in person months
- t_d = The nominal delivery time for the schedule
- t_d = Actual delivery time desired

Putnam Norden Rayleigh (PNR) Curve



t_n = Nominal delivery time

t_o = Optimal delivery time (in terms of cost) = $2\sqrt{n}$

FIRST TERM EXAMINATION [SEPT. 2015]
FIFTH SEMESTER [B. TECH]
SOFTWARE ENGINEERING [ETCS–303]

Time: 1½ Hrs. **MM : 30**

Note: *Q.No.1 is compulsory and attempt any two questions from the rest.*

Q.1. What is the root cause of failure in waterfall model? **(2)**

Ans. Studies have shown that in over 80% of the investigated and failed software projects, the usage of the Waterfall methodology was one of the key factors of failure.When deploying the waterfall methodology there is a strict sequential chain of the different project phases. A previous phase has to be completed before starting the next phase. Going back is in most cases difficult, costly, frustrating to the team and time consuming.

The project timeline is planned at the start. A releasable product is delivered only at the end of the project timeline. If one phase is delayed all other phases are also delayed.

Q.1. (b) List the characteristics of SRS document? **(2)**

Ans. An SRS should be:

Correct: An SRS is correct if, and only if, every requirement stated there in is one that the software shall meet. Traceability makes this procedure easier and less prone to error.

Unambiguous: An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.

Complete: An SRS is complete if, and only if, it includes the following elements:

All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.

Consistent: Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct. An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

Verifiable: An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.

Nonverifiable requirements include statements such as "works well", "good human interface", and "shall usually happen". These requirements cannot be verified because it is impossible to define the terms "good", "well", or "usually".

Modifiable: An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to

Have a coherent and easy-to-use organization with a table of contents, an index, and explicit crossreferencing;

Traceable: An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:

Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.

Forward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number.

Q.1. (c) What do you mean by a live variable. **(2)**

Ans. A variable is live from its first to its last reference within a process. The number of live variable represents the degree of difficulty of the ststement. The average number of live variablies is the sum of count of live variables divided by the number of executable statements.

Q.1. (d) Define the term Risk assessment. **(2)**

Ans. Risk assessment is the determination of quantitative or qualitative estimate of risk related to a well-defined situation and a recognized threat (also called hazard).a systematic process of evaluating the potential risks that may be involved in a projected activity or undertaking.

Risk identification is the first step in risk assessment, which identifies all the different risks for a particular project. These risks are project-dependent and identifying them is an exercise in envisioning what can go wrong. Methods that can aid risk identification include checklists of possible risks, surveys, meetings and brainstorming, and reviews of plans, processes, and work products

Q.1. (e) Justify the statement that every deliverable can be a milestone but every milestone is not nessarily a deliverable. **(2)**

Ans. Deliverable is a term used in project management to describe a tangible or intangible object produced as a result of the project that is intended to be delivered to a customer (either internal or external).A deliverable could be a report, a document, a server upgrade or any other building block of an overall project.

A deliverable may be composed of multiple smaller deliverables. It may be either an outcome to be achieved (as in "The corporation says that becoming profitable this year is a deliverable ") or an output to be provided (as in "The deliverable for the completed project consists of a special-purpose electronic device and its controlling software.").

A deliverable differs from a project milestone in that a milestone is a measurement of progress toward an output whereas the deliverable is the result of the process. For a typical project, a milestone might be the how to Calculate peak manning and average rate of software build up.completion of a product design while the deliverable might be the technical diagram of the product.

A deliverable is more than just a project document in that a project document is typically part of a project deliverable. A project deliverable may contain a number of documents and physical things.

In technical projects, deliverables can be further classified as hardware, software, or design documents. Deliverable may refer to an item specifically required by contract documents, such as an item on a Contract Data Requirements List or mentioned in the statement of work.

A deliverable is something (hard or soft) that can be ready to dispatch to the site or the Client as partial item of the supply foreseen in the contract. *e.g.* when the project has started some part of the design (when settled), can be anticipated to the sub-supplier who has therefore the possibility of starting his purchase activity of raw material, even if many other parameters are not yet designed by the designer.how to Calculate peak manning and average rate of software build up.

Q.2. What is the difference between SEI CMM model and ISO 9001.Why CMM is better than ISO 9001. **(5)**

Ans. Difference between ISO 9000 and CMM(ISO 9000 VS CMM)

ISO 900(INTERNATIONAL STANDARD ORGANISATION) and CMM (CABABILITY MATURITY MODEL)

It applies to any type of industry.

CMM

CMM is specially developed for software industry.

CMM focuses on the software Engineering activities.

CMM gets nto technical aspect of software engineering.

CMM has 5 levels:

Initial, Repeatable ,Defined , Managed, Optimization

Similarly other process in CMM are not included in ISO 9000

1. Project tracking
2. Process and technology change management
3. Intergroup coordinating to meet customer's requirements
4. Organization level process focus, process development and integrated management.

ISO 9000

ISO 9000 specifies minimum requirement.

ISO 9000 restricts itself to what is required. It suggests how to fulfill the requirements.

ISO 9000 provides pass or fail criteria.

It provides grade for process maturity.

ISO 9000 has no levels.

ISO 9000 does not specifies sequence of steps required to establish the quality system.

It reconnects the mechanism for step by step progress through its successive maturity levels.

Certain process elements that are in ISO are not included in CMM like:

1. Contract management
2. Purchase and customer supplied components
3. Personal issue management
4. Packaging ,delivery, and installation management

The Capability Maturity Model for Software (CMM), developed by the Software Engineering Institute, and the ISO 9000 series of standards, developed by the International Standards Organization, share a common concern with quality and process management. The two are driven by similar concerns and intuitively correlated. The purpose of this report is to contrast the CMM and ISO 9001, showing both their differences and their similarities. The results of the analysis indicate that, although an ISO 9001-compliant organization would not necessarily satisfy all of the level 2 key process areas, it would satisfy most of the level 2 goals and many level 3 goals. Because there are practices in the CMM that are not addressed in ISO 9000, it is possible for a level 1 organization to receive 9001 registration; similarly, there are areas addressed by ISO 9001 that are not addressed in the CMM. A level 3 organization would have little difficulty in obtaining ISO 9001 certification, and a level 2 organization would have significant advantages in obtaining certification.

Q.2. (b) What do you mean by Function Points?Discuss function point techniques **(5)**

Ans. A function point is a "unit of measurement" to express the amount of business functionality an information system (as a product) provides to a user. Function points are used to compute a functional size measurement (FSM) of software. The cost (in dollars or hours) of a single unit is calculated from past projects.

A Function Point (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user. FPs measure software size. They are widely accepted as an industry standard for functional sizing.

For sizing software based on FP, several recognized standards and/or public specifications have come into existence. As of 2013, these are "

COSMIC "ISO/IEC 19761: 2011 Software engineering. A functional size measurement method.

FiSMA "ISO/IEC 29881: 2008 Information technology - Software and systems engineering - FiSMA 1.1 functional size measurement method.

IFPUG "ISO/IEC 20926: 2009 Software and systems engineering - Software measurement - IFPUG functional size measurement method.

Mark-II " ISO/IEC 20968: 2002 Software engineering - Mk II Function Point Analysis - Counting Practices Manual.

NESMA " ISO/IEC 24570: 2005 Software engineering - NESMA function size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis. Object Management Group Specification for Automated Function Point

Function Point Analysis (FPA) technique quantifies the functions contained within software in terms that are meaningful to the software users. FPs consider the number of functions being developed based on the requirements specification.

Function Points (FP) Counting is governed by a standard set of rules, processes and guidelines as defined by the International Function Point Users Group (IFPUG). These are published in Counting Practices Manual (CPM).

Q.3. Assume a project which takes development time of 2.5 years with manpower requirement of 300PY.

(i) Calculate peak manning and average rate of software build up. **(3)**

(ii) What is the manpower cost after 1.5 years. **(2)**

Ans.

(i) Peak manning

$$m_0 = \frac{k}{td\sqrt{e}} = \frac{300\times10\times100}{2.5\times1.648}$$

$$= \frac{12000\times10}{1648} = 72.82$$

(ii) Av. rate of s/w team build up

$$= \frac{m_0}{td} = \frac{72}{2.5} = 28.8 \text{ person/year}$$

or

2.4 person/month

Q.3. (b) Discuss the differences between throw away and evolutionary prototyping. **(5)**

Ans. An evolutionary prototype is one that is built such that it can be expanded upon and revised, but does not have to be discarded and completely rewritten in order to go to market. A throw-away prototype is something that's designed to capture the "essence" of whatever it is that you're prototyping, but that will be completely replaced by something else that will be what goes to market.

The spiral model as defined by Barry Boehm consists of defining the requirements as much as possible, creating a design that helps you to identify risks and explore possible solutions, prototyping your design, and then producing a release. Once you release, you begin the process again to produce the next release.

There are two kinds of prototypes which are used:

A throwaway prototype is made quickly with the intention of discarding it after you have learned from it. It's not well designed nor well implemented. Your goal is to get something in front of you client so they can see it and respond to it in order to help you refine your requirements and move toward a system the client wants. However, this is risky since the client sees a system and might equate that with a working system - you need to stress that it's just a prototype.

In evolutionary prototype, you take more care when developing the prototype as you will be refactoring and expanding your prototype into the final product. You can actually deliver an evolutionary prototype to your client and have them use it just as they would the actual system - you slowly refine the prototype into a final product that is delivered.

Q.4. (a) Discuss Facilitated Application Specification technique.Explain how brainstorming is different from FAST **(6)**

Ans. Facilitated Application Specification Technique ("FAST")

- It is a technique for requirements elicitation for software development.
- The objective is to close the gap between what the developers intend and what users expect.
- It is a team-oriented approach for gathering requirements.

Basic guidelines

1. Meetings are conducted at a neutral site attended by both developers and users.
2. The group establishes rules for preparation and participation.
3. An agenda is suggested that with enough formality to cover all important points but informal enough to encourage the free flow of ideas.
4. A facilitator controls the meeting.
5. A definition mechanism is used.

- The main goal is to identify the problem, propose solutions, negotiate different approaches, and specify a preliminary set of software requirements in an atmosphere that is conductive to accomplish the goal.
- After initial meeting, user and developer should write a one or two product request form. Before the next meeting it is distributed to all other attendees. Each attendee is asked to make the following lists:

1. List of objects
2. List of services
3. List of constraints
4. performance criteria

Representatives of FAST

1. Marketing person
2. Software and hardware engineer
3. Representative from manufacturing
4. An outside facilitator

Q.4. (b) Discuss spiral model of software development. **(4)**

Ans. The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Advantages of Spiral model: High amount of risk analysis hence, avoidance of Risk is enhanced. Good for large and mission-critical projects. Strong approval and documentation control. Additional Functionality can be added at a later date. Software is produced early in the software life cycle.

Disadvantages of Spiral model: Can be a costly model to use. Risk analysis requires highly specific expertise. Project's success is highly dependent on the risk analysis phase. Doesn't work well for smaller projects.

When to use Spiral model: When costs and risk evaluation is important

For medium to high-risk projects

Long-term project commitment unwise because of potential changes to economic priorities

Users are unsure of their needs requirements are complex

New product line

significant changes are expected (research and exploration)

SECOND TERM EXAMINATION [NOV. 2015]
FIFTH SEMESTER [B. TECH]
SOFTWARE ENGINEERING [ETCS–303]

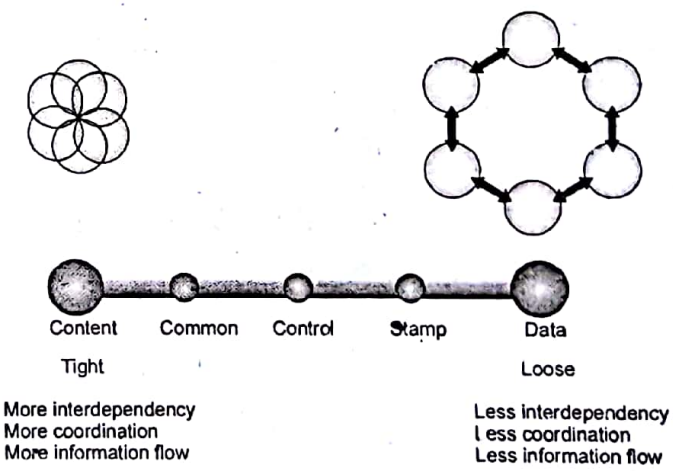
Time: 1½ Hrs. **MM : 30**

Note: *Q.No.1 is compulsory and attempt any two questions from the rest.*

Q.1. Define Coupling. **(2)**

Ans. Coupling refers to the degree to which software components are dependant upon each other. For instance, in a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled. In a loosely-coupled architecture, components can remain autonomous and allow middleware software to manage communication between them. In a decoupled architecture, the components can operate completely separately and independently.

Coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between modules.



Q.1. (b)Define the term Debugging? **(2)**

Ans. Debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code or the engineering of a hardware device. To debug a program or hardware device is to start with a problem, isolate the source of the problem, and then fix it. A user of a program that does not know how to fix the problem may learn enough about the problem to be able to avoid it until it is permanently fixed. When someone says they've debugged a program or "worked the bugs out" of a program, they imply that they fixed it so that the bugs no longer exist. Debugging is the process of finding and resolving of defects that prevent correct operation of computer software or a system. Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another.

Q.1. (c) What is reverse engineering? (2)

Ans. Software reverse engineering is the process of recovering the design and the requirement specification of a product from an analysis of its code.The purpose of reverse engineering is to facilitate.

Q.1. (d) What is software failure? How is it related with a fault?

Ans. Fault: It is a condition that causes the software to fail to perform its required function.

Error: Refers to difference between Actual Output and Expected output.Error is terminology of Developer.

Failure: It is the inability of a system or component to perform required function according to its specification.

Fault: An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. It is an inherent weakness of the design or implementation which might result in a failure. A fault might be present and latent in the systems like they were in Patriot Missile failure and Therac-25 accidents. These faults lead to a failure when the exact scenario is met.

Fault avoidance – using techniques and procedures which aim to avoid the introduction of faults during any phase of the safety lifecycle of the safety-related system

Fault tolerance – the ability of a functional unit to continue to perform a required function in the presence of faults or errors

Failure: The inability of a system or component to perform its required functions within specified performance requirements

Q.1. (e) Draw the bath tub curve of hardware reliability by naming its three phases. (2)

Ans. There are three phases i.e Burn-in,Useful Life, Wear-Out.

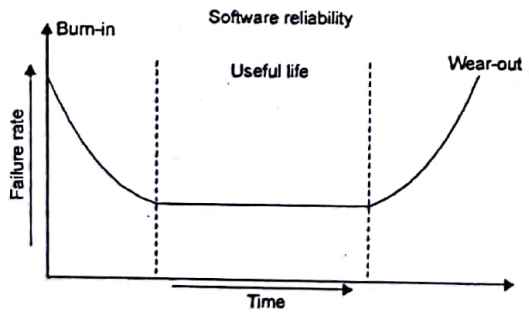


Fig. Bath tub curve of hardware reliability

In Burn-in phase, the failure rate is quite high initially and it starts decreasing gradually as the time progresses. It may be due to initial testing in the premises of the organization. During Useful life period, the failure rate is almost constant. Failure rate increases in the wear out phase due to wearing out or aging of the components.

The best period is Useful life period and the shape of this curve is like a bath tub thats why it is known as bath tub curve.

Q.2. (a) Define module cohesion and explain its various types. (6)

Ans. Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

Co-incidental cohesion - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

Logical cohesion - When logically categorized elements are put together into a module, it is called logical cohesion.

Temporal cohesion - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

Procedural cohesion - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion

Communicational cohesion - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.

Sequential cohesion - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.

Functional cohesion - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Q.2. (b) What is mutation testing? What is the purpose of mutation score. (4)

Ans. Mutation testing is a structural testing technique, which uses the structure of the code to guide the testing process. On a very high level, it is the process of rewriting the source code in small ways in order to remove the redundancies in the source code.These ambiguities might cause failures in the software if not fixed and can easily pass through testing phase undetected.

Mutation Testing Benefits: It brings a whole new kind of errors to the developer's attention. It is the most powerful method to detect hidden defects, which might be impossible to identify using the conventional testing techniques. Debugging and Maintaining the product would be more easier than ever.

Mutation testing is a method of software testing in which program or source code is deliberately manipulated, followed by suite of testing against the mutated code. The mutations introduced to source code are designed to imitate common programming errors. A good unit test suite typically detects the program mutations and fails automatically.

Mutation testing is used on many different platforms, including Java, C++, C# and Ruby. The goal of Mutation Testing is to assess the quality of the test cases which should be robust enough to fail mutant code. This method is also called as Fault based testing strategy as it involves creating fault in the program.

The mutation score is defined as the percentage of killed mutants with the total number of mutants.

Mutation Score = (Killed Mutants/Total number of Mutants) * 100

Test cases are mutation adequate if the score is 100%. Experimental results have shown that mutation testing is an effective approach for the measuring the adequacy of the test cases. But, the main drawback is that the high cost of generating the mutants and executing each test case against that mutant program.

Q.3. (a) Explain basic execution time model. (5)

Ans. The basic execution time model is based on an execution time model, that is, the time taken during modeling is the actual CPU execution time of the software being modeled. The model is simple to understand and apply, and its predictive value has been generally found to be good.

The model focuses on failure intensity while modeling reliability. It assumes that the failure intensity decreases with time, that is, as (execution) time increases, the failure intensity decreases. This assumption is generally true as the following is assumed about the software testing activity, during which data is being collected: during testing, if a failure is observed, the fault that caused that failure is detected and the fault is removed. Even if a specific fault removal action might be unsuccessful, overall failures lead to a reduction of faults in the software. Consequently, the failure intensity decreases. Most other models make a similar assumption, which is consistent with actual observations.

In the basic model, it is assumed that each failure causes the same amount of decrement in the failure intensity. That is, the failure intensity decreases with a constant rate with the number of failures. In the more sophisticated Musa's logarithmic model, the reduction is not assumed to be linear but logarithmic (this is the basic difference between the basic and logarithmic model). For the basic model, the failure intensity (number of failures per unit time) as a function of the number of failures is given as

$$l(\mu) = I_0(1-\mu/v_0),$$

where I_0 is the initial failure intensity at the start of execution (*i.e.*, at time $p = 0$), μ is the expected number of failures by the given time p and v_0 is the total number of failures that would occur in infinite time. The total number of failures in infinite time is finite as it is assumed that on each failure, the fault in the software is removed. As the total number of faults in a given software whose reliability is being modeled is finite, this implies that the number of failures is finite.

The linear decrease in failure intensity, as the number of failures observed increases is an assumption that is likely to hold for software for which the operational profile is uniform. That is, for software where the operational profile is such that any valid input is more or less equally likely, the assumption that the failure intensity decreases linearly generally holds. The intuitive rationale is that if the operational profile is uniform, any failure can occur at any time and all failures will have the same impact in failure intensity reduction. If the operational profile is not uniform, the failure intensity curves are ones whose slope decreases with the number of failures (i.e., each additional failure contributes less to the reduction in failure intensity). In such a situation, the logarithmic model is better suited.

Note that the failure intensity decreases due to the nature of the software development process, in particular system testing, the activity in which reliability modeling is applied. Specifically, during testing when a failure is detected, the fault that caused the failure is identified and removed. It is removal of the fault that reduces the failure intensity. However, if the faults were not removed, as would be the situation if the software were already deployed in the field (when the failures are logged or reported but the faults are not removed), then the failure intensity would stay constant. In this situation, the value of A would stay the same as at the last failure that resulted in fault removal, and the reliability will be given by

$R(p) = e$-lt, where t is the execution time.

Q.3. (b)Differentiate between function oriented design and object oriented design. (5)

Ans. FOD is function oriented design and OOD is 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: Functions are grouped together by which a higher level function is Page on obtained.an eg of this technique is SA/SD.

OOD: Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.

3. FOD: In this appproach 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.

4. 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.

5. FOD - we decompose in function/procedure level

OOD: - we decompose in class level

Q.4. (a) What do you mean by cyclomatic complexity.Describe all methods of calculating cyclomatic complexity by taking an example. (6)

Ans. Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:

1.-Cyclomatic complexity = E – N + P

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

Example:

```
IF A = 10 THEN
IF B > C THEN
    A = B
  ELSE
    A = C
  ENDIF
ENDIF
Print A
Print B
Print C
```

FlowGraph: The Cyclomatic complexity is calculated using the above control flow diagram that shows seven nodes(shapes) and eight edges (lines), hence the cyclomatic complexity is 8 – 7 + 2 = 3

Q.4. (b) Explain Boehm's maintenance model with the help of a diagram. (4)

Ans. In 1988, Barry Boehm proposed a more comprehensive life cycle model called the Spiral Model to address the inadequacies of the Waterfall Model. According to Boehm, "the major distinguishing feature of the Spiral Model is that it creates a risk-driven approach to the software process rather than a primarily document-driven or code-driven process. It incorporates many of the strengths of other models and resolves many of their difficulties" . Software projects encompass many different areas of risk such as project cost overruns, changed requirements (e.g., the DIA baggage system), loss of key project personnel, delay of necessary hardware, competition from other software developers, technological breakthroughs which obsolete the project, and many others. The essential concept of the Spiral Model is "to minimize risks by the repeated use of prototypes [emphasis added] and other means. The Spiral Model works by building progressively more complete versions of the software by starting at the center of the spiral and working outwards. With each loop of the spiral, the customer evaluates the work and suggestions are made for its modification. Additionally, with each loop of the spiral, a risk analysis is performed which results in a 'go / no-go' decision. If the risks are determined to be too great then the project is terminated". Thus, the Spiral Model addresses the problem of requirements engineering through development of prototypes, and it addresses the need for risk management by performing risk analysis at each step of the life cycle.

In order to manage the risk of a single phase in the Spiral Model (*i.e.*, one loop of the spiral), Boehm used the template below for risk assessment during the development of a software productivity tool. The rows of the template represented various management elements of the project. For each new phase, he created a new instance of the template to review the status of the project and decided whether the risks were too great to continue.

The more iterative your development process, the more times you spiral around

You spiral inward from the high-level descriptions down to the lower level implementation details (note: this directionality is inverted from Boehm – this model doesn't try to convey the amount of cost or work in each loop around the spiral)

As you spiral down, the activities change. "Design" at the high level might be on paper. But as you spiral down, design is about turning those paper documents into executable code. Same for the other quadrants.

Quadrants

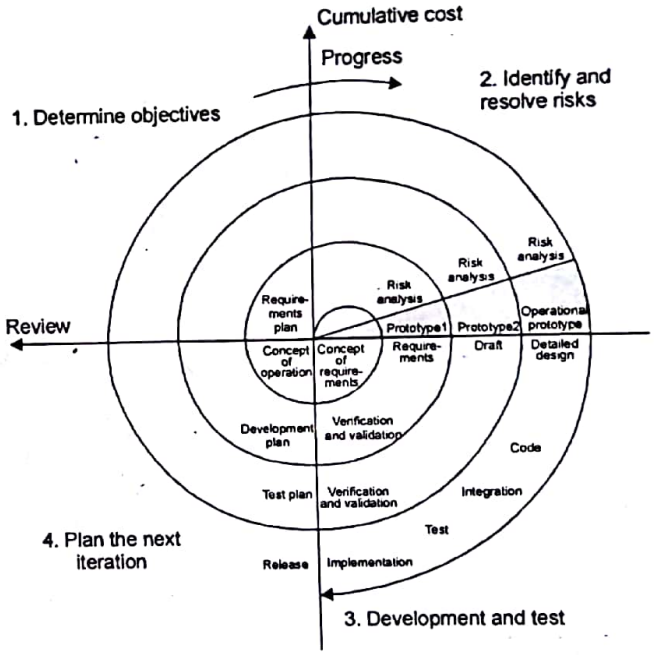
Customer - What does the customer think? In one of the better trends of the last 20 years since Boehm's paper, agile methodologies have recognized the customer as an essential direct participant of the development process.

Plan - What do we plan to do? This includes requirements analysis, priorities, risks, and schedules. At the very high level, it may be corporate goals. At the very low level, it might be writing an automated functional test before writing the code to make that test pass.

Design - How will we do it? At the high level, design is done via documents, diagrams, and discussion. At the lowest level, design is expressed as the executable code that constitutes the product.

Test - Have we done it right? At the high level, we discuss and review ideas and documents. At the lowest level, we execute tests against the functioning product.

The four quadrants align with how we tend to specialize our people and organizations as we grow. In a Microsoft organizational model, it aligns with customer, program management, development, and test.



END TERM EXAMINATION [DEC. 2015]
FIFTH SEMESTER [B. TECH]
SOFTWARE ENGINEERING [ETCS–303]

Time: 3 Hrs. **M.M : 75**

Note: *Attempt any five questions including Q. no. 1 which is compulsory.*

Q.1. (a) What do u understand by software engineering? Explain. (5×5=25)

Ans. Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Need of Software Engineering: The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

Cost- As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

Quality Management- Better process of software development provides better and quality software product.

Characteristics of good software: A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

Operational

Transitional

Maintenance

Q.1. (b) Differentiate between verification and validation

Ans.

Verification	Validation
1. Are we building the system right?	Are we building the right system?
2. Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	Validation is the process of evaluating software at the end of the development process to determine whether software meets the customer expectation and requirements.

3. The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.
4. Following activities are involved in Verification: Reviews, Meetings and Inspections.	Following activities are involved in Validation: Testing like black box testing, white box testing, gray box testing etc.
5. Verification is carried out by QA team to check whether implementation software is as per specification document or not.	Validation is carried out by testing team.
6. Execution of code is not comes under Verification.	Execution of code is comes under Validation.
7. Verification process explains whether the outputs are according to inputs or not.	Validation process describes whether the software is accepted by the user or not.
8. Verification is carried out before the Validation.	Validation activity is carried out just after the Verification.
9. Following items are evaluated during Verification: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during Validation: Actual product or Software under test.
10. Cost of errors caught in Verification is less than errors found in Validation.	Cost of errors caught in Validation is more than errors found in Verification.
11. It is basically manually checking the of documents and files like requirement specifications etc.	It is basically checking of developed program based on the requirement specifications documents and files.

Q.1. (c) Differentiate between path testing and mutation testing

Ans. Path Testing is a structural testing method based on the source code or algorithm and NOT based on the specifications. It can be applied at different levels of granularity.

Path Testing Assumptions:

(a) The Specifications are Accurate

(b) The Data is defined and accessed properly

(c)There are no defects that exist in the system other than those that affect control flow

Path Testing Techniques: Control Flow Graph (CFG) - The Program is converted into Flow graphs by representing the code into nodes, regions and edges.

Decision to Decision path (D-D) - The CFG can be broken into various Decision to Decision paths and then collapsed into individual nodes.

Independent (basis) paths - Independent path is a path through a DD-path graph which cannot be reproduced from other paths by other methods.

Mutation testing is a structural testing technique, which uses the structure of the code to guide the testing process. On a very high level, it is the process of rewriting the source code in small ways in order to remove the redundancies in the source code

These ambiguities might cause failures in the software if not fixed and can easily pass through testing phase undetected.

Mutation Testing Benefits:

(*a*) It brings a whole new kind of errors to the developer's attention.

(*b*) It is the most powerful method to detect hidden defects, which might be impossible to identify using the conventional testing techniques.

(c) Tools such as Insure++ help us to find defects in the code using the state-of-the-art.

(*d*) Increased customer satisfaction index as the product would be less buggy.

(*e*) Debugging and Maintaining the product would be more easier than ever.

Mutation Testing Types: Value Mutations: An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.

Decision Mutations: The decisions/conditions are changed to check for the design errors. Typically, one changes the arithmetic operators to locate the defects and also we can consider mutating all relational operators and logical operators (AND, OR , NOT)

Statement Mutations: Changes done to the statements by deleting or duplicating the line which might arise when a developer is copy pasting the code from somewhere else.

Q.1. (*d*)What is information flow metrics?Explain with example

Ans. Data flow, or information flow, means parameter passing and variable access.

There are several metrics for measuring the information flow: IFIN, IFOUT, IFIO and IC1.

Watching for procedures with high informational complexity can reveal the following issues: procedure has more than one function; procedure is a stress point in the system (with information traffic); procedure has excessive functional complexity. High informational complexity indicates candidate procedures for extensive testing or redesign.

Several metrics have been developed to measure information flow complexity. Generally speaking, two concepts are common to all information flow metrics: fan-in and fan-out. Fan-in is the information that flows into a procedure. Fan-out is what comes out of it. In addition, the concept of procedure length is used. The exact definition of what is in fan-in, fan-out and length may vary.

Project Analyzer traditionally uses the following formulas for calculations of information flow.

Fan-in IFIN = Procedures called + parameters read + global variables read. Informational fan-out (IFOUT) estimates the information a procedure returns. Fan-out IFOUT = Procedures that call this procedure + ByRef parameters written to + global variables written to IFIO Informational fan-in × fan-out

Fan-out indicate coupling between modules. High fan out means highly coupled module. High fan-out indicates module depends highly on other module and thus shows poor design structure.

High fan-out also increases maintainability cost. Any changes in module require seeing other modules and thus increases maintainability cast see in following example.

Q.1. (*e*) Differentiate between failure and faults

Ans. Fault : An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. It is an inherent weakness of the design or implementation which might result in a failure. A fault might be present and latent in the systems like they were in Patriot Missile failure and Therac-25 accidents. These faults lead to a failure when the exact scenario is met.

Fault avoidance – using techniques and procedures which aim to avoid the introduction of faults during any phase of the safety lifecycle of the safety-related system

Fault tolerance – the ability of a functional unit to continue to perform a required function in the presence of faults or errors

Failure: The inability of a system or component to perform its required functions within specified performance requirements

Error : A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

Q.2. (*a*) What is software crisis? And also discuss the process of software development. **(6.5)**

Ans. Software crisis is a term used for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could be tackled. With the increase in the complexity of the software, many software problems arose because existing methods were neither sufficient nor up to the mark.

The term "software crisis" was coined by some attendees at the first NATO Software Engineering Conference in 1968 at Garmisch, Germany.

The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

(*a*) Projects running over-budget
(*b*) Projects running over-time
(*c*) Software was very inefficient
(*d*) Software was of low quality
(*e*) Software often did not meet requirements
(*f*) Projects were unmanageable and code difficult to maintain
(*g*) Software was never delivered

The software development process describes the over-arching process of developing a software product. Sometimes referred to as the software lifecycle, this process may be used for the implementation of a single application or a far-reaching ERP system.

While there is not standard definition, most development processes include the following activities:

Requirement gathering
Design
Implementation
Testing
Maintenance

A software development process or life cycle is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

1. Requirement gathering and analysis: Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.

Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

2. Design: In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

In this phase the testers comes up with the Test strategy, where they mention what to test, how to test.

3. Implementation/Coding: On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

4. Testing: After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase all types of functional testing like unit testing, integration testing, system testing, acceptance testing are done as well as non-functional testing are also done.

5. Deployment: After successful testing the product is delivered / deployed to the customer for their use.

As soon as the product is given to the customers they will first do the beta testing. If any changes are required or if any bugs are caught, then they will report it to the engineering team. Once those changes are made or the bugs are fixed then the final deployment will happen.

6. Maintenance: Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

Q.2. (b) Describe the various software metrics. **(6)**

Ans. A software metric is a standard of measure of a degree to which a software system or process possesses some property. Even if a metric is not a measurement (metrics are functions, while measurements are the numbers obtained by the application of metrics), often the two terms are used as synonymous. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget

planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

Product Metrics: In software development process, a working product is developed at the end of each successful phase. Each product can be measured at any stage of its development. Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements. If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Measurement is done by metrics: Three parameters are measured: process measurement through process metrics, product measurement through product metrics, and project measurement through project metrics.

Product metrics help software engineer to detect and correct potential problems before they result in catastrophic defects. In addition, product metrics assess the internal product attributes in order to know the efficiency of the following:

Analysis, design, and code model. Potency of test cases.

Overall quality of the software under development. Various metrics formulated for products in the development process are listed below:

Metrics for analysis model: These address various aspects of the analysis model such as system functionality, system size, and so on.

Metrics for design model: These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.

Metrics for source code: These assess source code complexity, maintainability, and other characteristics.

Metrics for testing: These help to design efficient and effective test cases and also evaluate the effectiveness of testing.

Metrics for maintenance: These assess the stability of the software product.

Metrics for the Analysis Model: There are only a few metrics that have been proposed for the analysis model. However, it is possible to use metrics for project estimation in the context of the analysis model. These metrics are used to examine the analysis model with the objective of predicting the size of the resultant system. Size acts as an indicator of increased coding, integration, and testing effort; sometimes it also acts as an indicator of complexity involved in the software design. Function point and lines of code are the commonly used methods for size estimation.

Function Point (FP) Metric: The function point metric, which was proposed by A.J Albrecht, is used to measure the functionality delivered by the system, estimate the effort, predict the number of errors, and estimate the number of components in the system. Function point is derived by using a relationship between the complexity of software and the information domain value. Information domain values used in function point include the number of external inputs, external outputs, external inquires, internal logical files, and the number of external interface files.

Lines of Code (LOC): Lines of code (LOC) is one of the most widely used methods for size estimation. LOC can be defined as the number of delivered lines of code, excluding comments and blank lines. It is highly dependent on the programming language used as code writing varies from one programming language to another. For example, lines of

code written (for a large program) in assembly language are more than lines of code written in C++.

From LOC, simple size-oriented metrics can be derived such as errors per KLOC (thousand lines of code), defects per KLOC, cost per KLOC, and so on. LOC has also been used to predict program complexity, development effort, programmer performance, and so on. For example, Hasltead proposed a number of metrics, which are used to calculate program length, program volume, program difficulty, and development effort.

Metrics for Specification Quality:

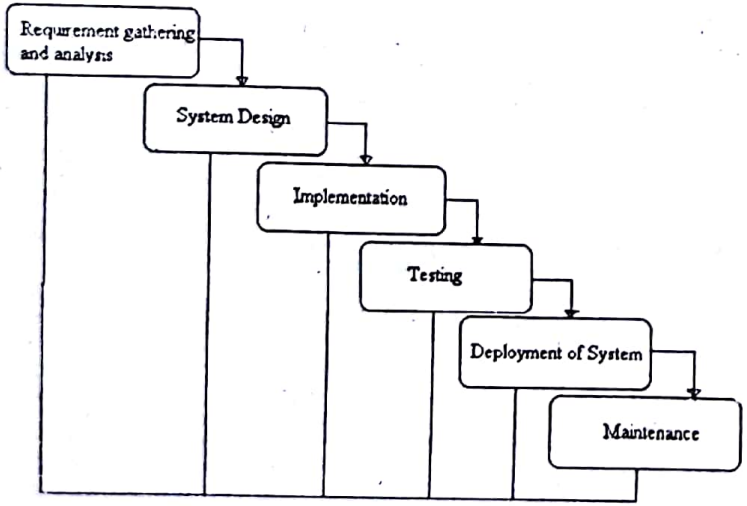
Project Metrics: Project metrics enable the project managers to assess current projects, track potential risks, identify problem areas, adjust workflow, and evaluate the project team's ability to control the quality of work products. Note that project metrics are used for tactical purposes rather than strategic purposes used by the process metrics.

Project metrics serve two purposes. One, they help to minimize the development schedule by making necessary adjustments in order to avoid delays and alleviate potential risks and problems. Two, these metrics are used to assess the product quality on a regular basis-and modify the technical issues if required. As the quality of the project improves, the number of errors and defects are reduced, which in turn leads to a decrease in the overall cost of a software project.

Q.3. (a) Differentiate between waterfall,prototype,evolutionary and spiral model. **(6)**

Ans. Types of Software developing life cycles (SDLC)

- Waterfall Model
- Evolutionary Prototyping Model
- Spiral Method (SDM)
- Iterative and Incremental Method



Several Overview of "Waterfall Model"

Waterfall Model

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model.** It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

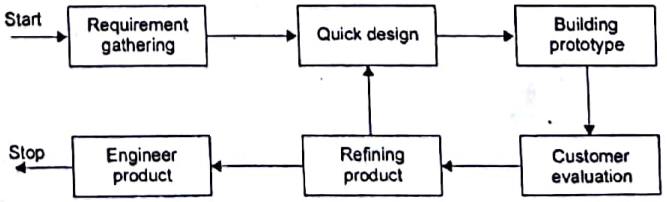
Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model - each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

Prototyping Model: The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.



Prototyping model

The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

Advantages of Prototype model: Users are actively involved in the development. Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.

Errors can be detected much earlier. Quicker user feedback is available leading to better solutions. Missing functionality can be identified easily Confusing or difficult functions can be identified Requirements validation, Quick implementation of, incomplete, but functional, application.

Disadvantages of Prototype model: Leads to implementing and then repairing way of building systems. Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis.

When to use Prototype model: Prototype model should be used when the desired system needs to have a lot of interaction with the end users.

Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.

Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Spiral Model: The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

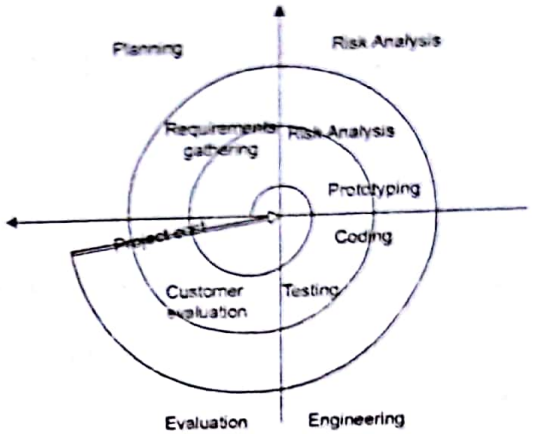
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

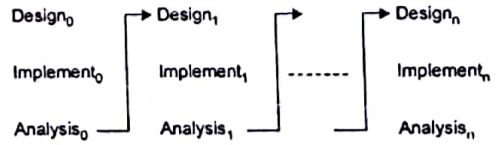
Diagram of Spiral model:



Advantages of Spiral model: High amount of risk analysis hence, avoidance of Risk is enhanced. Good for large and mission-critical projects. Strong approval and documentation control. Additional Functionality can be added at a later date. Software is produced early in the software life cycle.

Disadvantages of Spiral model: Can be a costly model to use. Risk analysis requires highly specific expertise. Project's success is highly dependent on the risk analysis phase. Doesn't work well for smaller projects

Evolutionary model: An iterative life cycle model or evolutionary model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.



In the diagram above when we work iteratively we create rough product or product piece in one iteration, then review it and improve it in next iteration and so on until it's finished. As shown in the image above, in the first iteration the whole painting is sketched roughly, then in the second iteration colors are filled and in the third iteration finishing is done. Hence, in iterative model the whole product is developed step by step.

Advantages of Iterative model: In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.

In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.

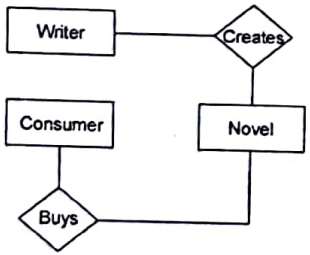
In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.

In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative model: Each phase of an iteration is rigid with no overlaps. Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

Q.3. (b) In which phase of the software development you draw ER-diagram?Explain with example **(4.5)**

Ans. ER diagrams are frequently used during the design stage of a development process in order to identify different system elements and their relationships with each other. For example, an inventory software used in a retail shop will have a database that monitors elements such as purchases, item, item type, item source and item price. Rendering this information through an ER diagram would be something like this:



In the diagram, the information inside the oval shapes are attributes of a particular entity.

Q.4. (a)What are behavioural and non-behavioural requirements. **(6)**

Ans. Functional Requirements: Most requirements definition focuses mainly on functional requirements, which are based upon the expected functioning of the product or system to be created. Functioning, typically is equated with product/system features for which you might have a menu or button choice, such as: identify a customer, select an item to order, and calculate the amount due.

All things considered, requirements definers probably are best at identifying functional requirements, although they often overlook and get wrong more of the functional requirements than they ordinarily recognize. On hindsight reflection, they frequently do realize that many of the problems which surface later, and thus are harder and more expensive to fix, are attributable to inadequately addressed nonfunctional requirements.

Nonfunctional Requirements

Nonfunctional requirements refer to a whole slew of attributes including performance levels, security, and the various "ilities," such as usability, reliability, and availability. Invariably, requirements definers get wrapped up in how the product/ system is expected to function and lose sight of these added elements.

When such factors are not addressed adequately, seemingly proper product/system functioning in fact fails to function. For example, a system may identify customers in such a slow, insecure, and difficult to use manner that it can cause mistakes which : make data unreliable, provoke frustration-based attempted work-around that can create further problems, and ultimately lead to abandonment.

That's the recognized way in which nonfunctional requirements inpact product/ system success. Other ofter often unrecognized issues also need to be appreciated.

Q.4. (b)Discuss the Putnam Resource Allocation model in detail

Ans. Putnam model describes the *time* and *effort* required to finish a software project of specified *size*. SLIM (Software LIfecycle Management) is the name given by Putnam to the proprietary suite of tools his company QSM, Inc. has developed based on his model. It is one of the earliest of these types of models developed, and is among the most widely used. Closely related software parametric models is Constructive Cost Model (COCOMO).

Putnam used his observations about productivity levels to derive the software equation:

$$\frac{B^{1/3}.\text{Size}}{\text{Productivity}} = \text{Effort}^{1/3}.\text{Time}^{4/3}$$

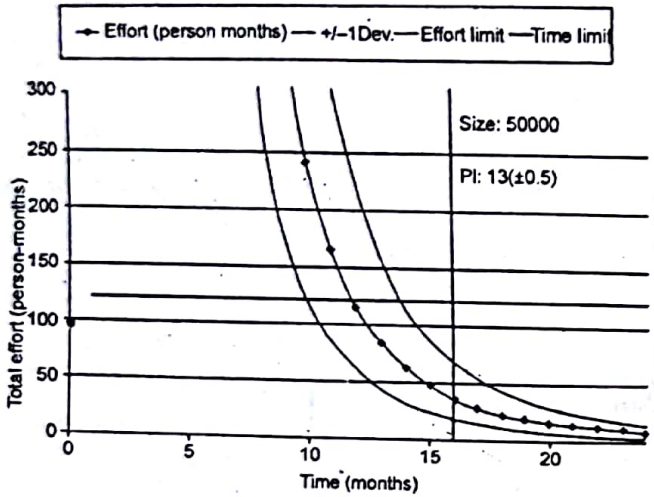
where:

- Size is the product size (whatever size estimate is used by your organization is appropriate). Putnam uses ESLOC (Effective Source Lines of Code) throughout his books.
- B is a scaling factor and is a function of the project size.
- Productivity is the **Process Productivity,** the ability of a particular software organization to produce software of a given size at a particular defect rate.
- Effort is the total effort applied to the project in person-years.
- Time is the total schedule of the project in years.

In practical use, when making an estimate for a software task the software equation is solved for effort:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity}.\ \text{time}^{4/3}}\right]^3.B$$

An estimated software size at project completion and organizational process productivity is used. Plotting *effort* as a function *of time* yields the *Time-Effort Curve.* The points along the curve represent the estimated total effort to complete the project at some *time.* One of the distinguishing features of the Putnam model is that total effort decreases as the time to complete the project is extended. This is normally represented in other parametric models with a schedule relaxation parameter.



This estimating method is fairly sensitive to uncertainty in both *size* and *process productivity*. Putnam advocates obtaining process productivity by calibration:

$$\text{Process Productivity} = \frac{\text{Size}}{\left[\frac{\text{Effort}}{B}\right]^{1/3} \cdot \text{Time}^{4/3}}$$

Putnam makes a sharp distinction between 'conventional productivity": *size I effort* and process productivity.

One of the key advantages to this model is the simplicity with which it is calibrated. Most software organizations, regardless of maturity level can easily collect *size, effort* and duration *(time)* for past projects. Process Productivity, being exponential in nature is typically converted to a linear *productivity index* an organization can use to track their own changes in productivity and apply in future effort estimates.

Q.5. (*a*) What are the software requirements and specification?Explain

Ans. A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

Qualities of SRS:

Correct

Unambiguous

Complete

Consistent

Ranked for importance and/or stability

Verifiable

Modifiable

Traceable

Q.5. (*b*) Differentiate between single and multivariate model. **(4.5)**

Ans. Multivariate regression is a technique that estimates a single regression model with more than one outcome variable. When there is more than one predictor variable in a multivariate regression model, the model is a multivariate multiple regression.

Q.6. (*a*) Discuss the cohesion and coupling and their types with examples. **(8)**

Ans.

Cohesion	Coupling
Cohesion is the indication of the relationship within module.	Coupling is the indication of the relationships between modules.
Cohesion shows the module's relative functional strength.	Coupling shows the relative independence among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the single thing.	Coupling is a degree to which a component/module is connected to the other modules.
While designing you should strive for high cohesion *i.e.* a cohesive component/ module focus on a single task (*i.e.*, single-mindedness) with little interaction with other modules of the system.	While designing you should strive for low coupling i.e. dependency between modules should be less.

Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.	Making private fields, private methods and non public classes provides loose coupling.
Cohesion is Intra – Module Concept.	Coupling is Inter -Module Concept.

Q.6. (b) If 99% of the program is written in FORTRAN and the remaining 1% in assembly language,the percentage increase in the programming time compared to writing the entire the entire program in FORTRAN and rewriting the 1% in assembly language? **(4.5)**

Ans. Explanation: The first case takes 99 + 10 = 109 man–day

The second case require 100 + 4 = 104 man-day

$$\therefore \quad \text{Percentage} = \frac{(109-104)\times 100}{100} = \frac{500}{100} = 5$$

Q.7. Write the short note. **(5 x 2.5 = 12.5)**

Ans. (a) Decision table testing: A decision table is a good way to deal with combinations of things (*e.g.* inputs). This technique is sometimes also referred to as a 'cause-effect' table. The reason for this is that there is an associated logic diagramming technique called 'cause-effect graphing' which was sometimes used to help derive the decision table.

Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.

Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.

It helps the developers to do a better job can also lead to better relationships with them. Testing combinations can be a challenge, as the number of combinations can often be huge. Testing all combinations may be impractical if not impossible. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important. If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

Q.7. (b) Cause Effect graphing

Ans. Cause Effect Graph is a black box testing technique that graphically illustrates the relationship between a given outcome and all the factors that influence the outcome.

It is also known as Ishikawa diagram as it was invented by Kaoru Ishikawa or fish bone diagram because of the way it looks.

Cause Effect - Flow Diagram

Cause Effect Graph

Circumstances - under which Cause-Effect Diagram used. To Identify the possible root causes, the reasons for a specific effect, problem, or outcome. To Relate the

interactions of the system among the factors affecting a particular process or effect. To Analyze the existing problems so that corrective action can be taken at the earliest.

Benefits: It Helps us to determine the root causes of a problem or quality using a structured approach.

It Uses an orderly, easy-to-read format to diagram cause-and-effect relationships. It Indicates possible causes of variation in a process. It Identifies areas, where data should be collected for further study. It Encourages team participation and utilizes the team knowledge of the process. It Increases knowledge of the process by helping everyone to learn more about the factors at work and how they relate.

Q.7. (c) Unit Testing

Ans. Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually.

Unit Testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed.

A unit is the smallest testable part of software. It usually has one or a few inputs and usually a single output. In procedural programming a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.)

Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

Q.7. (d)Integration Testing.

Ans. Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.

Also after integrating two different components together we do the integration testing. As displayed in the image below when two different modules 'Module A' and 'Module B' are integrated then the integration testing is done.

Integration testing is done by a specific integration tester or test team. Integration testing follows two approach known as 'Top Down' approach and 'Bottom Up' approach as shown in the image below:

1. Big Bang integration testing: In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole. As per the below image all the modules from 'Module 12 to 'Module 62 are integrated simultaneously then the testing is carried out. What is big bang integration testing

Advantage: Big Bang testing has the advantage that everything is finished before integration testing starts.

Disadvantage: The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.

2. Top-down integration testing: Testing takes place from top to bottom, following the control flow or architectural structure (*e.g.* starting from the GUI or main menu). Components or systems are substituted by stubs.

Q. 7. (e)System Testing

Ans. System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic.

System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system.

System testing falls under the black box testing category of software testing. White box testing is the testing of the internal workings or code of a software application. In contrast, black box or system testing is the opposite.System testing involves the external workings of the software from the user's perspective.

Q.8. (a) Discuss the various characteristics of good SRS. **(6)**

Ans. Software requirement specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it. The basic goal of the requirement phase is to produce the SRS, Which describes the complete behavior of the proposed software. SRS is also helping the clients to understand their own needs.

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. For example, the only aircraft that is totally safe is one that cannot be started, contains no fuel or other liquids, and is securely tied down.

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. The real world environment is not well known for new applications and for mature applications the real world keeps changing. The *Y2K* problem with the transition from the year 1999 to the year 2000 is an example of the real world moving beyond an application's specified requirements.

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.

Q.8. (b) What are the roles of documentation in software life cycle model. **(6.5)**

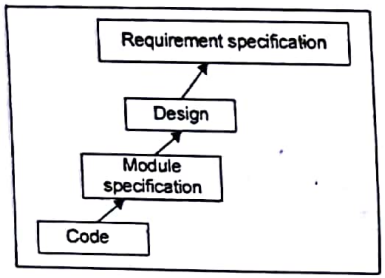
Ans. If we observe then the fact is, projects that have all the documents have high level of maturity. Most companies do not give even a little importance to the documentation as much they give to software development process. If we search on web then we can find various templates on how to create various types of documents. But how many of them are really used by organizations or individuals?

Fact is that, careful documentation can save an organization's time, efforts and money. While going for any type of certification, why there is stress given on documentation, it's just because it shows importance of client and processes to individual and organization. Unless you are able to produce document that is comfortable to user no matter how good your product is, no one is going to accept it.

It's my experience, we own one product, which is having a bit confusing functionality. When I started working on that I asked for some help documents to Manager and I got answer "No, we don't have any documents" Then I made an issue of that, because as a QA I knew, no one can understand how to use the product without documents or training. And if user is not satisfied, how we are going to make money out of that product?

"Lack of documentation is becoming a problem for acceptance" – Wietse Venema. Even same thing is applicable for User manuals. Take an example of Microsoft, they launch every product with proper documents, even for Office 2007 we have such documents, which are very explanatory and easy to understand for any user. That's one of the reasons that all their products are successful.

Maintenance work by improning the undestandafility of a system and to produce the necessary documents for a legacy system. Reverse engineering is important since lagacy software products lack. Proper documentation and are highly unstructured.



A process model for reverse engineering.

FIFTH SEMESTER [B.TECH.]
FIRST TERM EXAMINATION [SEPT. 2016]
SOFTWARE ENGINEERING [ETCS–303]

Time : 1.30 hrs. **M.M. : 30**

Note: *Q.1 is compulsory and attempt any two more questions.*

Q.1.(a) Write the name of the phase of software life cycle for which IEEE recommended standard IEEE 830-1993 is used. **(2)**

Ans. IEEE recommended standard IEEE 830-1993 is used in first phase of SDLC i.e. Requirement Analysis. Software Development Life Cycle begins with Requirement Analysis phase, where the stakeholders discuss the requirements of the software that needs to be developed to achieve a goal. The aim of the requirement analysis phase is to capture the detail of each requirement and to make sure everyone understands the scope of the work and how each requirement is going to be fulfilled.Depending on which software development methodology is used, different approaches are taken in moving from one phase to another. For example, in the waterfall or V model, the requirement analysis phase are saved in a SRS (Software Requirement Specification) document and needs to be finalized before the next phase can take place.

Q.1.(b) Define Stakeholders. **(2)**

Ans. Stakeholders in any organization can represent group, a person or even another organization that might have indirect or direct stake to such organization. Stakeholders can affect any organization's objectives, actions or policies, and stakeholders can have different responsibilities, and considerations within such organization. In the past, stakeholders were representing a common conception that business can fundamentally rely upon, and has its effect of the economic capital of any organization. In today's business world, the stakeholders represent people; and infrastructure that are necessary to any organization where the interests of such organization can be protected by them.

Stakeholders can be categorized as Connected, Internal and External Stakeholders:

- Internal Stakeholders – represent employees, managers, and others that can affect the running of the organization on a daily basis.
- Connect Stakeholders – represent suppliers, shareholders, customers, and parties that might be investing in the business.
- External Stakeholders – represent the group that are not directly linked to the organization, but can influence the activities within the organization.

Q.1.(c) Which mode in Intermediate COCOMO represents complex products? **(2)**

Ans. Complex Products are represented in Embedded mode of Intermediate COCOMO. A software project that must be developed within a set of tight hardware, software and operational constraints. The equation for the Effort (E) and Development time (D) for this model are :

$$\text{Effort} = 3.6 * (\text{KLOC})^{1.20}$$

$$\text{Development time} = 2.5 * (\text{KLOC})^{0.32*}$$

Q.1.(d) Write the full form of FAST and QFD techniques. **(2)**

Ans. Refer Q 8. (a) of End Term Exam. 2016.

Q.1.(e) Explain the bath tub curve of hardware reliablity. (2

Ans. Refer Q. 8. (b) of End Term Exam. 2016

Q.2. (a) Discuss the prototyping model. What is the effect of designing a prototyping on the overall cost of the project. (5

Ans. Refer Q. 7. (c) of End Term Exam. 2016

Q.2. (b) For a program with number of unique operators n1=30 and number of unique operands n2=50, compute the following : (5

(i) Program volume **(ii) Effort and time**

(iii) Program length **(iv) Program level**

Ans. In this program total no. of operators and operands are not given. So let us assume that

N1=40(assumed)

N2=60(assumed)

n1=30(given)

n2=50(given)

Now,

Length(in terms of total no. of tokens) = N1+N2 = 40+60 =80

Estimated Length(in terms of unique operators and operands)

= n1*logn1 + n2*logn2

= 30*log30 + 50*log50

= 30*4.9 + 50*5.64

= 147+282

= 429

Program Volume(V) = Length * $\log_2$(n1+n2)

= 429*log(30+50)

= 429*log(80)

= 429*6.32

= 2711

Potential minimal volume=V* =(2+N2)log(2+N2)

= 62log(62)

= 62*5.95

=1558.9

Program level = 4* V*/V

= 4*1558.9/2711

= 2.3

Program Effort(E)=V^2/V*

= 4714 person hours

Time = E/S

= 4714/180

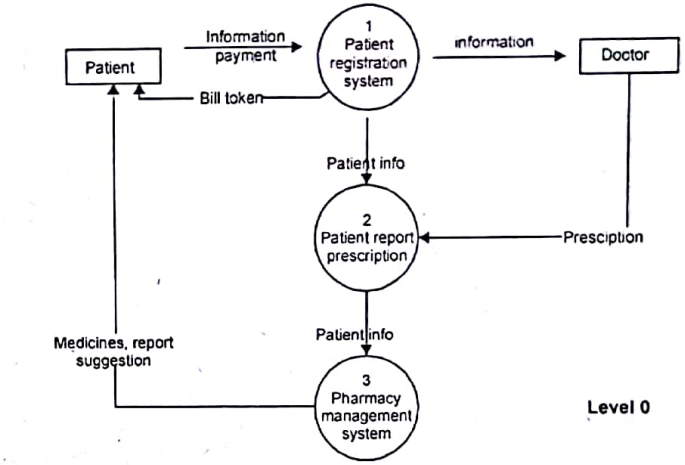
= 26 hours

Q.3. (a) Draw level 0 and level 1 DFD for Hospital Management System. (4)

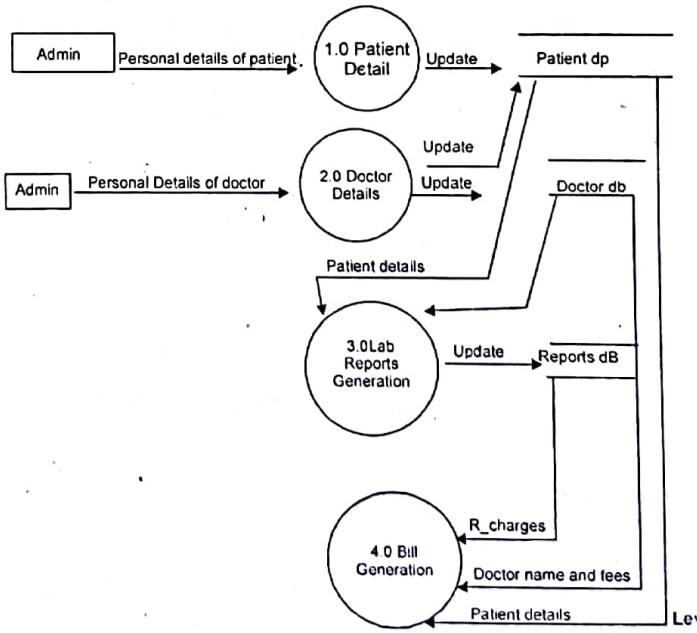
Ans. A context diagram is a top level (also known as Level 0) data flow diagram. It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities. In level 0 dfd, system is shown as one process.

The Level 0 DFD shows how the system is divided into 'sub systems' (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job and shows the flow of data between the various parts of the system.

0 level DFD for hospital management system



Level 0



Level 1

Q.3. (b) Explain Walston and Felix model. (2)

Ans. Refer Q .7. (b) of End Term of 2016

Q.3. (c) Discuss basic COCOMO model by discussing organic, semi-detached and embedded model. (4)

Ans. COCOMO (Constructive Cost Estimation Model) was proposed by Boehm [1981]. According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

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 lines of code (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 (hardware, software, operational, ...)

The basic COCOMO estimation model is given by the following expressions:

Effort Applied (E) = a *(KLOC)^b [man-months]

Development Time (D) = c*(Effort Applied)^d [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 ab, bb, cb and db are given in the following table (note: the values listed below are from the original analysis, with a modern reanalysis[4] producing different values):

Software project	a	b	c	d
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.

Q.4.(a) What are information flow metrics? Explain the basic information flow model. (4)

Ans. Cyclomatic complexity and the structural fan-in/fan-out metrics deal with the control flow. However, they don't take data flow into account. Data flow, or information flow, means parameter passing and variable access.

There are several metrics for measuring the information flow: IFIN, IFOUT, IFIO and IC1. Several metrics have been developed to measure information flow complexity. Generally speaking, two concepts are common to all information flow metrics: fan-in and fan-out. Fan-in is the information that flows into a procedure. Fan-out is what comes out of it. In addition, the concept of procedure length is used. The exact definition of what is in fan-in, fan-out and length may vary.

Project Analyzer traditionally uses the following formulas for calculations of information flow.

Fan-in IFIN = Procedures called + parameters read + global variables read

Informational fan-out (IFOUT) estimates the information a procedure returns.

Fan-out IFOUT = Procedures that call this procedure + ByRef parameters written to + global variables written to

IFIO Informational fan-in × fan-out

From IFIN and IFOUT, one can calculate a new metric: informational fan-in × fan-out.

IFIO = IFIN * IFOUT

IFIO is reportedly good in predicting the effort needed for implementing a procedure, but not so good at measuring complexity.

IC1 Informational complexity

Project Analyzer's formula to calculate informational complexity is as follows:

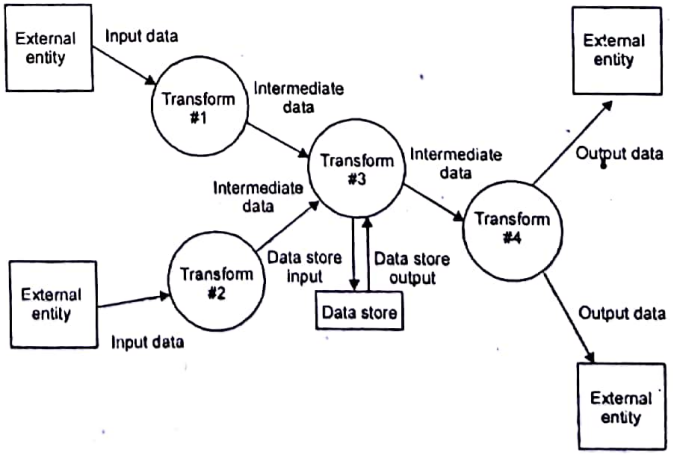
IC1 = LLOC * IFIO

As you can see, for IC1 we use the fan-in and fan-out values and multiply them by procedure length. We have defined length as follows:

length = LLOC = logical lines of code

Information Flow Model

Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in a variety of forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system. Output may light a single LED or produce a 200-page report. In effect, we can create a flow model for any computer-based system, regardless of size and complexity.



Structured analysis began as an information flow modeling technique. A computer-based system is represented as an information transform. A rectangle is used to represent an external entity; that is, a system element (e.g., hardware, a person, another program) or another system that produces information for transformation by the software or receives information produced by the software. A circle (sometimes called a bubble) represents a process or transform that is applied to data (or control) and changes it in some way. An arrow represents one or more data items (data objects). All arrows on a data flow diagram should be labelled. The double line represents a data store—stored information that is used by the software. The simplicity of DFD notation is one reason why structured analysis techniques are widely used.

It is important to note that no explicit indication of the sequence of processing or conditional logic is supplied by the diagram. Procedure or sequence may be implicit in the diagram, but explicit logical details are generally delayed until software design. It is important not to confuse a DFD with the flowchart.

Q.4. (b) Differentiate between functional and non-functional requirements. (3)

Ans. Functional requirements

Any requirement which specifies what the system should do. In other words, a functional requirement will describe a particular behaviour of function of the system when certain conditions are met, for example: "Send email when a new customer signs up" or "Open a new account".

A functional requirement for an everyday object like a cup would be: "ability to contain tea or coffee without leaking".

Typical functional requirements include:

Business Rules, Transaction corrections, adjustments and cancellations, Administrative functions, Authentication Authorization levels, Audit Tracking, External Interfaces, Certification Requirements, Reporting Requirements, Historical Data, Legal or Regulatory Requirements

Non-functional requirements

Any requirement which specifies how the system performs a certain function. In other words, a non-functional requirement will describe how a system should behave and what limits there are on its functionality.

Non-functional requirements generally specify the system's quality attributes or characteristics, for example: "Modified data in a database should be updated for all users accessing it within 2 seconds."

A non-functional requirement for the cup mentioned previously would be: "contain hot liquid without heating up to more than 45 °C".

Typical non-functional requirements include:

Performance, Scalability, Capacity, Availability, Reliability, Recoverability, Maintainability, Serviceability, Security, Regulatory, Manageability, Environmental, Data Integrity, Usability

Interoperability

It is important to correctly state non-functional requirements since they'll affect your users' experience when interacting with the system.

Q.4. (c) What is risk exposure? What techniques can be used to control each risk. (3)

Ans. A risk is a potential problem. It's an activity or event that may compromise the success of a software development project. Risk is the possibility of suffering loss,

and total risk exposure to a specific project will account for both the probability and the size of the potential loss.

An example would be that team is working on a project and the developer walks out of project and other person is recruited n his place and he doesn't work on the same platform and converts it into the platform he is comfortable with. Now the project has to yield the same result in the same time span. Whether they will be able to complete the project on time. That is the risk of schedule . Risk provides an opportunity to develop the project better. There is a difference between a Problem and Risk. Problem is some event which has already occurred but risk is something that is unpredictable.

Risk exposure= Size (loss)* probability of (loss)

Risk Management

Risk management means risk containment and mitigation. First, you've got to identify and plan. Then be ready to act when a risk arises, drawing upon the experience and knowledge of the entire team to minimize the impact to the project.

Risk management includes the following tasks:

- Identify risks and their triggers
- Classify and prioritize all risks
- Craft a plan that links each risk to a mitigation
- Monitor for risk triggers during the project
- Implement the mitigating action if any risk materializes
- Communicate risk status throughout project

1. Identify and Classify Risks

Most software engineering projects are inherently risky because of the variety potential problems that might arise. Experience from other software engineering projects can help managers classify risk. The importance here is not the elegance or range of classification, but rather to precisely identify and describe all of the real threats to project success. A simple but effective classification scheme is to arrange risks according to the areas of impact.

Five Types of Risk In Software Project Management

For most software development projects, we can define five main risk impact areas:

- New, unproven technologies
- User and functional requirements
- Application and system architecture
- Performance
- Organizational

2. Risk Management Plan

After cataloguing all of the risks according to type, the software development project manager should craft a risk management plan. As part of a larger, comprehensive project plan, the risk management plan outlines the response that will be taken for each risk—if it materializes.

3. Monitor and Mitigate

To be effective, software risk monitoring has to be integral with most project activities. Essentially, this means frequent checking during project meetings and critical events

(a) Monitoring includes:

- Publish project status report and include risk management issues.
- Revise risk plans according to any major changes in project schedule.
- Review and reprioritize risks, eliminating those with lowest probability.
- Brainstorm on potentially new risks after changes to project schedule or scope.

Mitigating options include:

- Accept: Acknowledge that a risk is impacting the project. Make an explicit decision to accept the risk without any changes to the project. Project management approval is mandatory here.
- Avoid: Adjust project scope, schedule, or constraints to minimize the effects of the risk.
- Control: Take action to minimize the impact or reduce the intensification of the risk.
- Transfer: Implement an organizational shift in accountability, responsibility, or authority to other stakeholders that will accept the risk.
- Continue Monitoring: Often suitable for low-impact risks, monitor the project environment for potentially increasing impact of the risk.

4. Communicate

Throughout the project it's vital to ensure effective communication among all stakeholders, managers, developers, QA-especially marketing and customer representatives. Sharing information and getting feedback about risks will greatly increase the probability of project success.

FIFTH SEMESTER [B.TECH.]
END TERM EXAMINATION [DECEMBER 2016]
SOFTWARE ENGINEERING [ETCS–303]

Time : 3 hrs. **M.M. : 75**

Note: *Attempts any five questions including Q.No. 1 which is compulsory.*

Q.1. Short answer type: **(5x5 =25)**

Q.1. (a) Give the features of Software characteristics. Draw the plot to indicate failure rate and time.

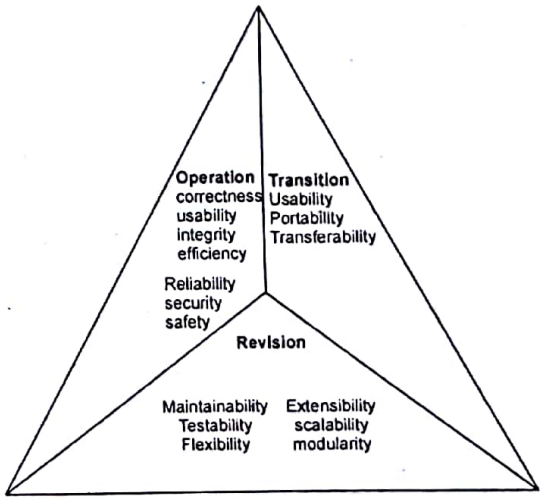
Ans. While developing any kind of software product, the first question in any developer's mind is, "What are the qualities that a good software should have ?"

The three characteristics of good application software are:

(1) Operational Characteristics

(2) Transition Characteristics

(3) Revision Characteristics



Operational Characteristics of software

These are functionality based factors and related to 'exterior quality' of software. Various Operational Characteristics of software are :

(a) **Correctness:** The software which we are making should meet all the specifications stated by the customer.

(b) **Usability/Learnability:** The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people.

(c) **Integrity :** Just like medicines have side-effects, in the same way a software may have a side-effect i.e. it may affect the working of another application. But a quality software should not have sideeffects.

(*d*) **Reliability :** The software product should not have any defects. Not only this, it shouldn't fail whilexecution.

(*e*) **Efficiency :** This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.

(*f*) **Security :** With the increase in security threats nowadays, this factor is gaining importance. The software shouldn't have ill effects on data/hardware. Proper measures should be taken to keep data secure from external threats.

(*g*) **Safety :** The software should not be hazardous to the environment/life.

Revision Characteristics of software

These engineering based factors of the relate to 'interior quality' of the software like efficiency, documentation and structure. These factors should be in-build in any good software. Various Revision Characteristics of software are:

(*a*) **Maintainability :** Maintenance of the software should be easy for any kind of user.

(*b*) **Flexibility :** Changes in the software should be easy to make.

(*c*) **Extensibility :** It should be easy to increase the functions performed by it.

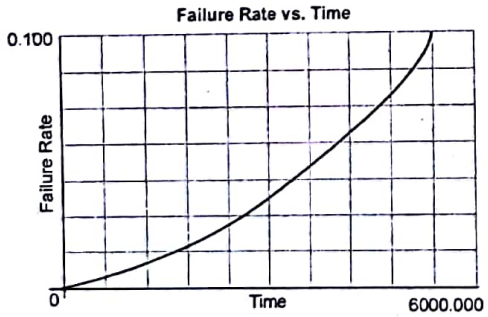
(*d*) **Scalability :** It should be very easy to upgrade it for more work(or for more number of users).

(*e*) **Testability :** Testing the software should be easy.

(*f*) **Modularity :** Any software is said to made of units and modules which are independent of each other. These modules are then integrated to make the final software. If the software is divided into separate independent parts that can be modified, tested separately, it has high modularity.

Transition Characteristics of the software :

(*a*) **Interoperability :** Interoperability is the ability of software to exchange information with other applications and make use of information transparently.



(*b*) **Reusability :** If we are able to use the software code with some modifications for different purpose then we call software to be reusable.

(*c*) **Portability :** The ability of software to perform same functions across all environments and platforms, demonstrate its portability.

Q.1. (*b*) Distinguish between failure and fault. **(5)**

Ans. Fault : An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. It is an inherent weakness of the design or implementation which might result in a failure.. These faults lead to a failure when the exact scenario is met.

Fault avoidance – using techniques and procedures which aim to avoid the introduction of faults during any phase of the safety lifecycle of the safety-related system

Fault tolerance – the ability of a functional unit to continue to perform a required function in the presence of faults or errors

Fault is a stage of software which is caused by an error/bug/defect/mistake. The fault is basically the original cause of an error. An uninitialized pointer variable in a C language program is an example of software faults.

Failure: It is a condition that causes the software to fail to perform its required function. In other words, Failure means "External behaviour is incorrect". If under certain environment and situation defects in the application or products are executed then system will produce the wrong results causing a failure.

Failure can also be caused because of other reasons like:

(1) Because of the environmental conditions as well like a radiation burst, a strong magnetic field, electronic field or pollution could cause faults in hardware. The faults might prevent or change the execution of software.

(2) Because of human error in interacting with the software perhaps wrong input value being entered or output being misinterpreted.

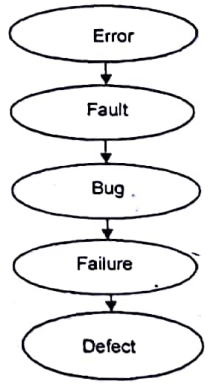
(3) Someone deliberately trying to cause a failure in the system

Example

Failures are caused by environment or sometime due to mishandling of product. Suppose we are using a compass just beside a current running wire then this will not show the correct direction and this is not helping in getting the right information from the product. In other way when a defect is found by end-user then this is called failure.

Failure occurs when fault executes. Hence the process of failure manifestation can therefore be represented as a behaviour chain as follows:

fault ->error ->failure



Relationship between error, fault, failure and defect

Thus,

1. A fault is a mistake or error caused by misjudgement, carelessness, and forgetfulness while a failure is the condition or state of not being able to meet an intended objective.

2. A fault is a character weakness, a frailty, or a shortcoming that can result in failures if not addressed well.

3. Failures can either be good or bad, but faults always have negative connotations.

4. To help oneself achieve success, it is very important to realize the circumstances that led to one's failure and to correct any faults that have caused it.

Q.1. (*c*) Compare functional and behavioural analysis models. **(5)**

Ans. Functional Model

Functional Modelling gives the process perspective of the object-oriented analysis model and an overview of what the system is supposed to do. It defines the function of the internal processes in the system with the aid of Data Flow Diagrams (DFDs). It depicts the functional derivation of the data values without indicating how they are derived when they are computed, or why they need to be computed.

A function model, similar with the activity model or process model, is a graphical representation of an enterprise's function within a defined scope. The purposes of the function model are to describe the functions and processes, assist with discovery of information needs, help identify opportunities, and establish a basis for determining product and service costs

Functional Modelling is represented through a hierarchy of DFDs. The DFD is a graphical representation of a system that shows the inputs to the system, the processing upon the inputs, the outputs of the system as well as the internal data stores. DFDs illustrate the series of transformations or computations performed on the objects or the system, and the external controls and objects that affect the transformation.

Rumbaugh et al. have defined DFD as, "A data flow diagram is a graph which shows the flow of data values from their sources in objects through processes that transform them to their destinations on other objects."

The four main parts of a DFD are:

Processes, Data Flows, Actors, and Data Stores.

The other parts of a DFD are:

Constraints, ControlFlows.

Behavioural Model

These are used to describe the overall behaviour of the system. Two types of behavioural model are :

- Data Processing models that shows how data is processed as it moves through the system.
- State machine models that shows system response to events.

These models show different perspective so both of them are required to describe the system behaviour. All behavioural models really do is describe the control structure of a system. This can be things like: Sequence of operations, Object states and Object interactions

Furthermore, this modelling layer can also be called Dynamic Modelling. The activity of creating a behavioural model is commonly known as behavioural modelling. As well as this, a system should also only have one behavioural model – much like functional modelling.

Behavioural models are represented with dynamic diagrams. For example:

(Design) Sequence Diagrams, Communication Diagrams or collaboration diagram, State Diagrams or state machine diagram or state chart

If we have both a sequence diagram AND and communication diagram, then together these are known as interaction diagrams, this is because they both represent how objects interact with one another using messages.

A behavioural model describes when the system is changing. The key feature (subject) of a behavioural model is – Objects.

Q.1. (d) Write short note on Spiral model. Also discuss its merits and demerits. (5)

Ans. The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

Spiral Model - Design

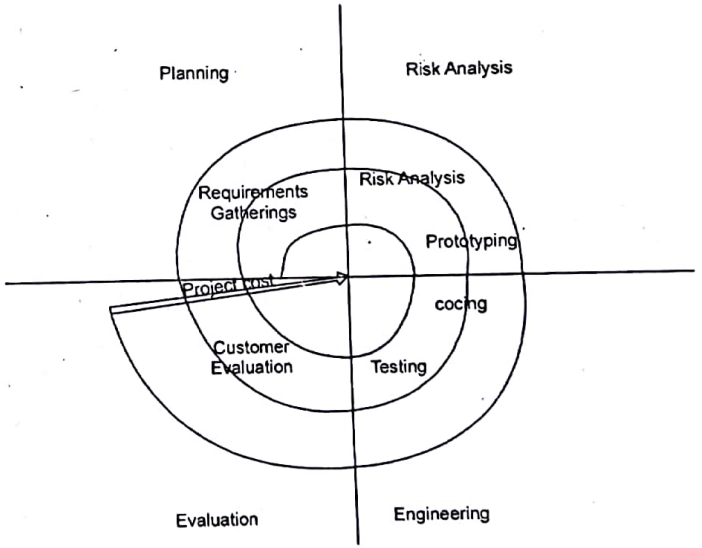
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Business Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.



Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

Q.1. (e) What are the responsibilities and challenges of software engineers? **(5)**

Ans. A Software engineer is a person responsible for the effective operation, maintenance and up-gradation of business software's used in organizations for effective business solutions.

Responsibilities of Software engineers:

- Documents and demonstrates solutions by developing documentation, flowcharts, layouts, diagrams, charts, code comments and clear code.
- Prepares and installs solutions by determining and designing system specifications, standards, and programming.
- Improves operations by conducting systems analysis; recommending changes in policies and procedures.
- Obtains and licenses software by obtaining required information from vendors; recommending purchases; testing and approving products.
- Updates job knowledge by studying state-of-the-art development tools, programming techniques, and computing equipment; participating in educational opportunities; reading professional publications; maintaining personal networks; participating in professional organizations.
- Protects operations by keeping information confidential.
- Provides information by collecting, analyzing, and summarizing development and service issues.
- Accomplishes engineering and organization mission by completing related results as needed.
- Develops software solutions by studying information needs; conferring with users; studying systems flow, data usage, and work processes; investigating problem areas; following the software development lifecycle.
- Train users to use new or modified equipment.
- Utilize microcontrollers to develop control signals, implement control algorithms and measure process variables such as temperatures, pressures and positions.
- Recommend purchase of equipment to control dust, temperature, and humidity in area of system installation.

- Train users to use new or modified equipment.
- Utilize microcontrollers to develop control signals, implement control algorithms and measure process variables such as temperatures, pressures and positions.
- Recommend purchase of equipment to control dust, temperature, and humidity in area of system installation.

Challenges of software engineers are :

- The methods used to develop small or medium-scale projects are not suitable when it comes to the development of large-scale or complex systems.
- Changes in software development are unavoidable. In today's world, changes occur rapidly and accommodating these changes to develop complete software is one of the major challenges faced by the software engineers.
- The advancement in computer and software technology has necessitated for the changes in nature of software systems. The software systems that cannot accommodate changes are not of much use. Thus, one of the challenges of software engineering is to produce high quality software adapting to the changing needs within acceptable schedules. To meet this challenge, the object oriented approach is preferred, but accommodating changes to software and its maintenance within acceptable cost is still a challenge.
- Informal communications take up a considerable portion of the time spent on software projects. Such wastage of time delays the completion of projects in the specified time.
- The user generally has only a vague idea about the scope and requirements of the software system. This usually results in the development of software, which does not meet the user's requirements.
- Changes are usually incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.
- The development of high-quality and reliable software requires the software to be thoroughly tested. Though thorough testing of software consumes the majority of resources, underestimating it because of any reasons deteriorates the software quality.

Q.2. (a) What are the characteristics to be considered for the selection of the life cycle model? **(6)**

Ans. The software process model framework is specific to the project. Thus, it is essential to select the software process model according to the software which is to be developed. The software project is considered efficient if the process model is selected according to the requirements. It is also essential to consider time and cost while choosing a process model as cost and/ or time constraints play an important role in software development. The basic characteristics required to select the process model are project type and associated risks, requirements of the project, and the users.

One of the key features of selecting a process model is to understand the project in terms of size, complexity, funds available, and so on. In addition, the risks which are associated with the project should also be considered. Note that only a few process models emphasize risk assessment. Various other issues related to the project and the risks are listed in Table.

Table Selections on the Basis of the Project Type and Associated Risks

Project Type and Associated Risks	Waterfall	Prototype	Spiral	RAD	Formal Methods
Reliability requirements	No	No	Yes	No	Yes
Stable funds	Yes	Yes	No	Yes	Yes
Reuse components	No	Yes	Yes	Yes	Yes
Tight project schedule	No	Yes	Yes	Yes	No
Scarcity of resources	No	Yes	Yes	No	No

The most essential feature of any process model is to understand the requirements of the project. In case the requirements are not clearly defined by the user or poorly understood by the developer, the developed software leads to ineffective systems. Thus, the requirements of the software should be clearly understood before selecting any process model. Various other issues related to the requirements are listed in Table.

Table Selection on the Basis of the Requirements of the Project

Requirements of the Project	Waterfall	Prototype	Spiral	RAD	Formal Methods
Requirements are defined early in SDLC	Yes	No	No	Yes	No
Requirements are easily defined and understandable	Yes	No	No	Yes	Yes
Requirements are changed frequently	No	Yes	Yes	No	Yes
Requirements indicate a complex System	No	Yes	Yes	No	No

Software is developed for the users. Hence, the users should be consulted while selecting the process model. The comprehensibility of the project increases if users are involved in selecting the process model. It is possible that a user is aware of the requirements or has a rough idea of the requirements. It is also possible that the user wants the project to be developed in a sequential manner or an incremental manner (where a part is delivered to the user for use). Various other issues related to the user's satisfaction are listed in Table.

Table Selection on the Basis of the Users

User Involvement	Waterfall	Prototype	Spiral	RAD	Formal Methods
Requires Limited User Involvement	Yes	No	Yes	No	Yes
User participation in all phases	No	Yes	No	Yes	No
No experience of participating in similar projects	No				

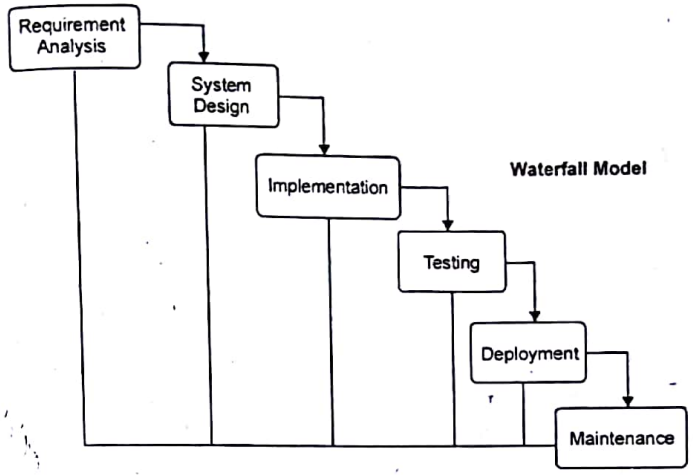
Q.2. (b) Mention at least two reasons as to why classical waterfall model can be considered impractical and cannot be used in real projects. (6.5)

Ans. Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially

Some situations where the use of Waterfall model is most appropriate are:

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.

- Ample resources with required expertise are available to support the product.
- The project is short.



Waterfall Model

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. The engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Thus, it is considered impractical and cannot be used in real projects because of following reasons:

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges earl

Q.3. (a) What are the different components of SRS document? (4)

Ans. A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

Qualities of SRS:

• Correct, Unambiguous, Complete, Consistent, Ranked for importance and/or stability, Verifiable, Modifiable, Traceable

• Conceptually, any SRS should have these components. Now we will discuss them one by one.

1. Functional Requirements

Functional requirements specify what output should be produced from the given inputs. So they basically describe the connectivity between the input and output of the system. For each functional requirement:

1. A detailed description of all the data inputs and their sources, the units of measure, and the range of valid inputs be specified:

2. All the operations to be performed on the input data obtain the output should be specified, and

3. Care must be taken not to specify any algorithms that are not parts of the system but that may be needed to implement the system.

4. It must clearly state what the system should do if system behaves abnormally when any invalid input is given or due to some error during computation. Specifically, it should specify the behaviour of the system for invalid inputs and invalid outputs.

2. Performance Requirements (Speed Requirements)

• This part of an SRS specifies the performance constraints on the software system. All the requirements related to the performance characteristics of the system must be clearly specified. Performance requirements are typically expressed as processed transaction s per second or response time from the system for a user event or screen refresh time or a combination of these. It is a good idea to pin down performance requirements for the most used or critical transactions, user events and screens.

3. Design Constraints

• The client environment may restrict the designer to include some design constraints that must be followed. The various design constraints are standard compliance, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

• **Standard Compliance:** It specifies the requirements for the standard the system must follow. The standards may include the report format and according procedures.

• **Hardware Limitations**: The software needs some existing or predetermined hardware to operate, thus imposing restrictions on the design. Hardware limitations can includes the types of machines to be used operating system availability memory space etc.

• **Fault Tolerance**: Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive, so they should be minimized.

• **Security:** Currently security requirements have become essential and major for all types of systems. Security requirements place restriction s on the use of certain commands control access to database, provide different kinds of access, requirements

for different people, require the use of passwords and cryptography techniques, and maintain a log of activities in the system.

4. External Interface Requirements

• For each external interface requirements:

1. All the possible interactions of the software with people hardware and other software should be clearly specified,

2. The characteristics of each user interface of the software product should be specified and

3. The SRS should specify the logical characteristics of each interface between the software product and the hardware components for hardware interfacing.

Q.3. (b) What are the benefits of ERD? (4)

Ans. ERD stands for Entity Relationship Diagram and it is basically a snapshot or summary of various data structures. ERD is designed to show the entities present in a database as well as the relationship between tables in that database. Learn the benefits of ERD use here below.

1. Visual representation: The foremost and most important ERD benefit is that it provides a visual representation of the design. It is normally crucial to have an ERD if you are looking to come up with an effective database design. This is because the patterns assist the designer in focusing on the way the database will primarily work with all the data flows and interactions. It is common to the ERD being used together with data flow diagrams so as to attain a better visual representation.

2. Effective communication: An ERD clearly communicates the key entities in a certain database and their relationship with each other. ERD normally uses symbols for representing three varying kinds of information. Diamonds are used for representing the relationships, ovals are usually used for representing attributes and boxes represent the entities. This allows a designer to effectively communicate what exactly the database will be like.

3. Simple to understand: ERD is easy to understand and simple to create. In effect, this design can be used to be shown to the representatives for both approval and confirmation. The representatives can also make their contributions to the design, allowing the possibilities of rectifying and enhancing the design.

4. High flexibility: The ERD model is quite flexible to use as other relationships can be derived easily from the already existing ones. This can be done using other relational tables and mathematical formulae.

The ERD thereby acts like the blueprint for the database and it allows the creation of an accurate design that reflects the needs of the project.

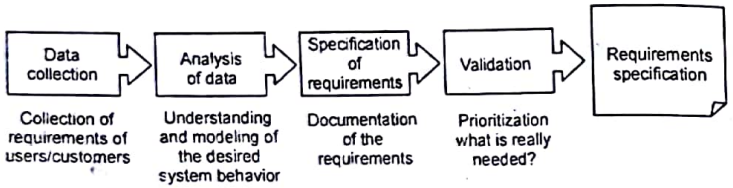
5. Easy conversion for E-R to other data model: Conversion from E-R diagram to a network or hierarchical data model can easily be accomplished.

Q.3. (c) What are the objectives of Requirements Analysis? Different types of Requirements. (4.5)

Ans. Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is an important aspect of project management. Requirements analysis involves frequent communication with system users to determine specific feature expectations, resolution

of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish. Energy should be directed towards ensuring that the final system or product conforms to client needs rather than attempting to mold user expectations to fit the requirements.

Requirements analysis is a team effort that demands a combination of hardware, software and human factors engineering expertise as well as skills in dealing with people.



Software Requirements

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product.Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples:

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include –

Security, Logging, Storage, Configuration, Performance, Cost, Interoperability, Flexibility, Disaster recovery, Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Q.4 (a) Write a note on Software Quality Asurance(SQA)? (4)

Ans. Software Quality Assurance (SQA) is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products).

It includes the following activities:

- Process definition and implementation
- Auditing
- Training

Processes could be:

- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing
- etc

Once the processes have been defined and implemented, Quality Assurance has the following responsibilities:

- identify weaknesses in the processes
- correct those weaknesses to continually improve the process

The quality management system under which the software system is created is normally based on one or more of the following models/standards:

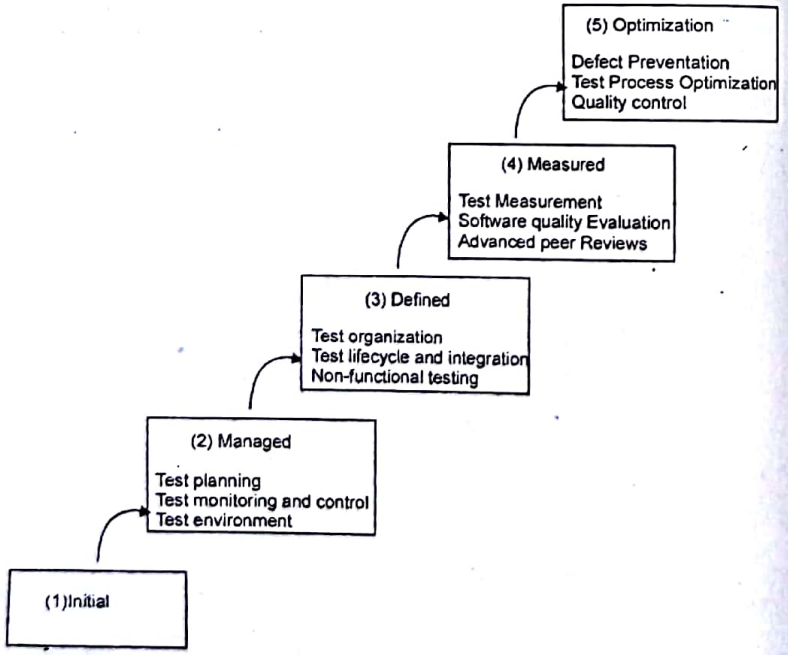
- CMMI
- Six Sigma
- ISO 9000

Software Quality Assurance encompasses the entire software development life cycle and the goal is to ensure that the development and/or maintenance processes are continuously improved to produce products that meet specifications/requirements. The process of Software Quality Control (SQC) is also governed by Software Quality Assurance (SQA).

SQA helps ensure the development of high-quality software. SQA practices are implemented in most types of software development, regardless of the underlying software development model being used. In a broader sense, SQA incorporates and implements software testing methodologies to test software. Rather than checking for quality after completion, SQA processes test for quality in each phase of development until the software is complete. With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards. SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies.

Q.4. (*b*) Compare ISO and SEI-CMM models. **(4)**

Ans. ISO 9000 is a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements to be implemented to maintain an efficient quality system. They are not specific to any one industry and can be applied to organizations of any size.



ISO 9000 can help a company satisfy its customers, meet regulatory requirements, and achieve continual improvement. However, it should be considered to be a first step, the base level of a quality system, not a complete guarantee of quality.

The Software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

Levels of CMM

ISO 900 (International Standard Organsiation)	CMM (Capability Maturity Model)
It applies to any type of industry.	CMM is specially developed for software industry
ISO 9000 addresses corporate business process	CMM focuses on the software Engineering activities.
ISO 9000 specifies minimum requirement.	CMM gets to technical aspect of software engineering.
ISO 9000 restricts itself to what is required.	It suggests how to fulfill the requirements.
ISO 9000 provides pass or fail criteria.	It provides grade for process maturity.
ISO 9000 has no levels.	CMM has 5 levels: Initial Repeatable Defined Managed Optimization
ISO 9000 does not specifies sequence of steps required to establish the quality system.	It reconnects the mechanism for step by step progress through its successive maturity levels.
Certain process elements that are in ISO are not included in CMM like: 1. Contract management 2. Purchase and customer supplied components 3. Personal issue management 4. Packaging, delivery, and installation management	Similarly other process in CMM are not included in ISO 9000 1. Project tracking 2. Process and technology change management 3. Intergroup coordinating to meet customer's requirements 4. Organization level process focus, process development and integrated management.

Q. 4. (*c*) How do you define Reliability? Discuss various models for reliability allocation **(4.5)**

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

- It is the probability of a failure free operation of a program for a specified time in a specified environment.
- It is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined no of input cases assuming that the hardware & the inputs are free of error. Hence, it is the probability that the software will work without failure for a specified period of time.

Software reliability models

To model software reliability , one must consider the principal factors that affect it fault introduction, fault removed & the environment.

• A Software reliability model specifies the general form of the dependence of the failure process on the above mentioned factors

(1) Basic Execution Time Model

• It is based on execution time

• It is assumed that failures may occur according to a non-homogenous poisson process (NHPP)

• In this model, the decrease in failure intensity, as a function of the no of failure observed is constant & in given as-

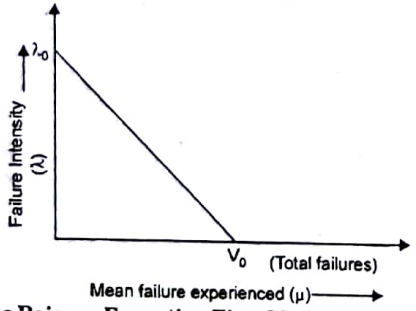
$$\lambda(\mu) = \lambda_0\left[1 - \frac{\mu}{V_0}\right]$$

Where

λ_0 - Initial failure intensity at the start of execution

V_0 - No. of failures experienced

μ - Avg. or expected no. of failure experienced at given point in time

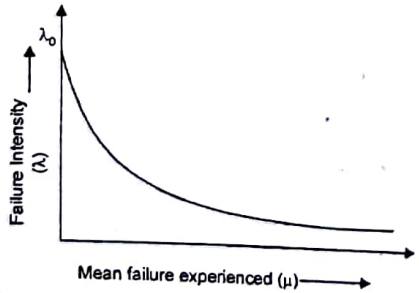


(2) Logarithmic Poisson Execution Time Model

• In this case, failure intensity function (decrement per failure) decreases exponentially, whereas it is constant for basic model

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

Where θ is failure intensity decay parameter



(3) Jelinski-Moranda Model

• It is best known reliability model

• It proposed a failure intensity function in the form of -

$$\lambda(t) = \phi(N-i+1)$$

Where, ϕ - constant of proportionality

N - Total no. of errors present

i - No. of errors found by time interval t_i

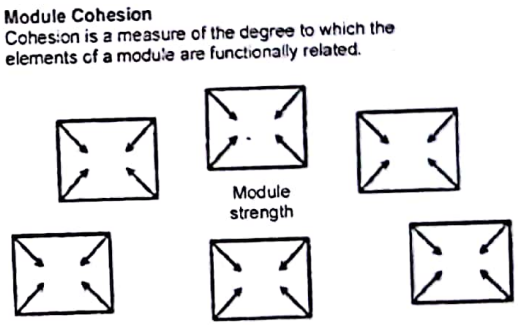
– This model assumes that all failures have the same failure rate

– So, every faliure contributes equally to the overall reliability.

– Here, failure intensity is directly proportional to no. of remaining errors in a program.

Q.5. (a) Define the term coupling and Cohesion? Explain various types of Coupling and cohesion? How are these concepts useful in arriving a good design of a system? **(8)**

Ans.



Cohesion can be defined as the degree of the closeness of the relationship between its components. In general, it measures the relationship strength between the pieces of functionality within a given module in the software programming. It is an ordinal type of measurement, which is described as low cohesion or high cohesion.

In a good module, the various parts having high cohesion is preferable due to its association with many desirable traits of the software such as reliability, reusability, robustness and understandability. On the other hand, a low cohesion is associated with the undesirable traits, including difficulty in maintaining, reusing and understanding. If the functionalities embedded in a class have much in common, then the cohesion will be increased in a system.

Cohesion has close relation with the coupling, which is completely a different concept. Low cohesion often correlates with the loose coupling and vice versa. Here are some advantages of the high cohesion

High cohesion leads to the increased module reusability because the developers of the application will easily find the component they look for in the cohesive set of operations offered by the module.

The system maintainability will be increased due to logical changes in the domain effecting fewer modules. The module complexity also reduces, when there is a high cohesion in the programming.

Types of Cohesion:

There are many different types of cohesion in the software engineering. Some of them are worst, while some of them are best. We have defined them below:

- Coincidental cohesion
- logical cohesion
- Temporal cohesion
- Procedural cohesion
- Communicational cohesion
- Sequential cohesion
- Functional cohesion

Coincidental cohesion: A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all. In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any thought or design. For example, in a transaction processing system (TPS), the get-input, print-error, and summarize-members functions are grouped into one module. The grouping does not have any relevance to the structure of the problem.

Logical cohesion: A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc. An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.

Temporal cohesion: When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion. The set of functions responsible for initialization, start-up, shutdown of some process, etc. exhibit temporal cohesion.

Procedural cohesion: A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.

Communicational cohesion: A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.

Sequential cohesion: A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next. For example, in a TPS, the get-input, validate-input, sort-input functions are grouped into one module.

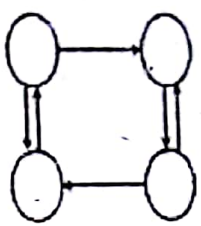
Functional cohesion: Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function. For example, a module containing all the functions required to manage employees' pay-roll exhibits functional cohesion. Suppose a module exhibits functional cohesion and we are asked to describe what the module does, then we would be able to describe it using a single sentence.

COUPLING

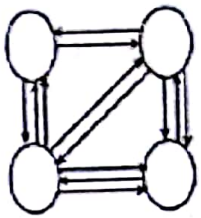
In software engineering, the coupling can be defined as the measurement to which the components of the software depend upon each other. Normally, the coupling is contrasted with the cohesion. If the system has a low coupling, it is a sign of a well-structured computer system and a great design. A low coupling combined with the high cohesion, it supports the mission of high readability and maintainability. The coupling term generally occurs together with the cohesion very frequently.

The coupling is an indication of the strength of the interconnection between all the components in a system. The highly coupled systems have interconnections, in which the program units depend upon each other, whereas in the loosely coupled systems

made up of components, that are independent of each other and have no dependence on each other.



Loosely coupled
some dependencies

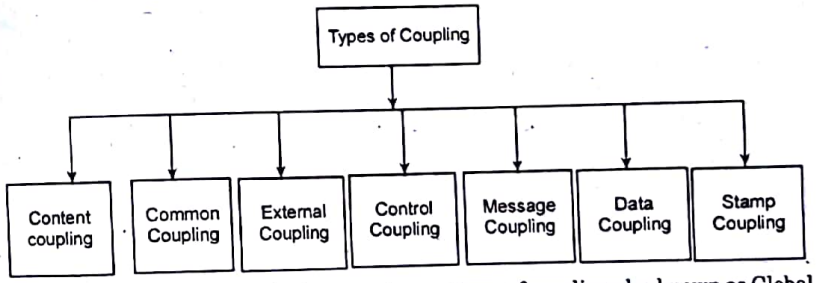


Highly coupled:
many dependencies

Types of Coupling:

The coupling has many types and can be high or low:

Content Coupling: Content Coupling is the highest type of coupling which occurs when one of the module relies on the other module's internal working. It means a change in the second module will lead to the changes in the dependent module.



Common Coupling: It is the second highest type of coupling also known as Global Coupling. It occurs when the same global data are shared by the two modules. In this, the modules will undergo changes if there are changes in the shared resource.

External Coupling: This type of coupling occurs when an external imposed data format and communication protocol are shared by two modules. External Coupling is generally related to the communication to external devices.

Control Coupling: In this type of coupling, one module controls the flow of another and passes information from one to another.

Message Coupling: This type of coupling can be achieved by the state decentralization. It is the loosest type of coupling, in which the component communication is performed through message passing.

Data Coupling: The modules are connected by the data coupling, if only data can be passed between them.

Stamp Coupling: In this type of coupling, the data structure is used to transfer information from on component to another.

Q.5. (*b*) Explain with examples top-down and bottom-up approach in software design. **(4.5)**

Ans. Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet

It may have the following steps involved:

• A solution design is created from requirement or previous used system and/or system sequence diagram.

• Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.

• Class hierarchy and relation among them is defined.

• Application framework is defined.

Software Design Approaches

Here are two generic approaches for software designing:

Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their on set of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.

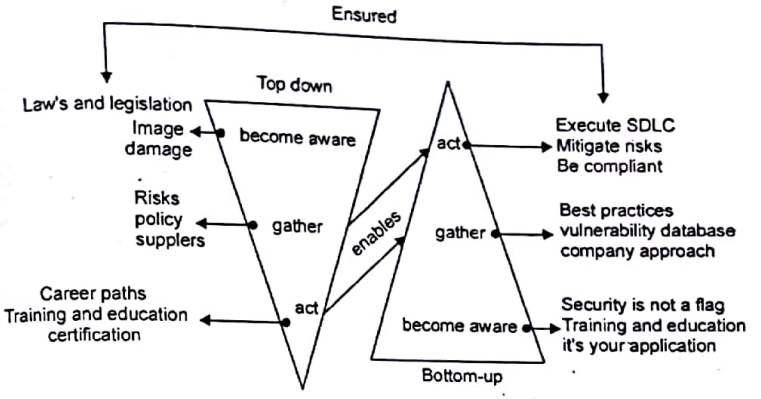
Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown. A top-down model is often specified with the assistance of "black boxes", these make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model. Top down approach starts with the big picture. It breaks down from there into smaller segments.

Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased. Bottom-up strategy is more suitable when a system needs to be created from some existing system, here the basic primitives can be used in the newer system.

Information enters the eyes in one direction (input), and is then turned into an image by the brain that can be interpreted and recognized as a perception (output). In a bottom-up approach the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a "seed" model, whereby the beginnings are small but eventually grow in complexity and completeness. However, "organic strategies" may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose.

For example, if an iterative enhancement type of process is being followed, in later iterations, the bottom-up approach could be more suitable (in the first iteration a top down approach can be used).



Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

Q.6.(a) What is software maintainability? How do you measure maintainability. **(4)**

Ans. Software maintainability is defined as the degree to which an application is understood, repaired, or enhanced. It is a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities and optimization. Software maintainability is important because it is approximately 75% of the cost related to a project. Finding ways to measure this important factor eases developer effort, decreases costs, and frees up resources. Understanding software maintainability allows organizations to identify improvement areas as well as determine the value supplied by current applications or during development changes.

Software maintainability requires more developer effort than any other phase of the development life cycle. A programming team will perform four types of maintenance on new deployments or enhancements: corrective, adaptive, perfective, and preventative. These activities will take additional time to complete if the code is not easy to manage in the first place.

Software maintenance activities can be classified as:

• Corrective maintenance – costs due to modifying software to correct issues discovered after initial deployment (generally 20% of software maintenance costs)

• Adaptive maintenance – costs due to modifying a software solution to allow it to remain effective in a changing business environment (25% of software maintenance costs)

• Perfective maintenance – costs due to improving or enhancing a software solution to improve overall performance (generally 5% of software maintenance costs)

• Enhancements – costs due to continuing innovations (generally 50% or more of software maintenance costs)

How to measure maintainability ?

The following steps should be undertaken to assess maintainability statically:

• A list of maintainability factors to be included in the assessment should be devised e.g. structure, complexity.

• Each factor (or group of factors) should be assigned a weighting to indicate its importance to the overall maintainability of the system. Each factor will have a maximum score of 10. The higher the score the less maintainable the system.

• During the assessment a score is awarded against each factor on the list. For example, a relatively old system may be awarded a score of 8 out of 10 to indicate that due to its age the system will relatively difficult to maintain.

• The scores for each of the factors assessed are then multiplied by the appropriate weighting and the resultant products are then summed to give an overall score which forms the Maintainability Measure of the system (the lower the score, the better the maintainability of the software system).

Example factors which can be used in a maintainability assessment are given below; the list is not exhaustive and should be modified to suit an individual organisation (although it is helpful if the same list is used throughout the organisation so comparisons between systems can be made):

Size, Maintainers' perception, Complexity, Environmental facilities, Structure, Maintenance relationships, Development process, System users/customers, Documentation, Maintenance team, Development team, Test facilities, Development timescale, Operating procedures, Maintenance procedures, Problem change traffic, Development relationships, Business change traffic

Using the Results

Once the Maintainability Measure has been derived, the weighted score for each factor assessed can be ranked to identify areas most in need of attention. For instance, action plans to address these areas, addressing the factors with the highest score first, can then be produced, detailing activities with the associated cost and maintainability benefit.

There are no objective (preset) targets for the Maintainability Measure; targets are set individually for each new system. Constraints such as time, cost, resource availability and technical limitations will affect the required Maintainability Measure.

Maintainability tests are considered to have passed when the Maintainability Measure from the assessment is such that the objectives can be met.

Q.6. (b) What is Boehm's cost estimation model for software maintenance. Explain. **(4)**

Ans. Maintenance effort is very significant and consumes about 40-70% of the cost of the entire life cycle. Thus, it is advisable to invest more effort in early phases of software life cycle to reduce the maintenance costs.

Boehm proposed a formula for estimating maintenance costs as part of his COCOMO model. Using data gathered from several projects, this formula was established in trms of effort. Boehm used a quantity called Annual Change Traffic(ACT) which is defined as :

The fraction of a software product's source instructions which udergo change during a year either through addition, deletion or modification.

The ACT is clearly related to the number of change requests.

$$ACT=(KLOC_{added} + KLOC_{deleted})/KLOC_{total}$$

The Annual Maintenance Effort(AME) in person-months can be calculated as:

AME=ACT X SDE

SDE : Software development effort in person-months

ACT : Annual Change Traffic

Suppose a software project required 400 person-months of development effort and it was estimated that 25% of the code would be modified in a year. The AME will be 0.25 X 400= 100 person_months. Boehm suggested that this rough estimate should be refined by judging the importance of factors affecting the cost and selecting the appropriate cost multipliers. Using these factors, Effort Adjustment Factor (EAF) should be calculated as in the case of COCOMO model.The modified equation is given below:

AME=ACT*SDE*EAF

Q.6. (c) Give the diffeence between Re-engineering and Reverse engineering. **(4.5)**

Ans. Re-engineering is commonly, but incorrectly, used in reference to reverse engineering. While both refer to the further investigation or engineering of finished products, the methods of doing so, and the desired outcomes, are vastly different. Reverse engineering attempts to discover how something works, while re-engineering seeks to improve a current design by investigating particular aspects of it.

Re-Engineering

Re-engineering is the investigation and redesign of individual components. It may also describe the entire overhaul of a device by taking the current design and improving certain aspects of it. The aims of re-engineering may be to improve a particular area of performance or functionality, reduce operational costs or add new elements to a current design. The methods used depend on the device but typically involve engineering drawings of the amendments followed by extensive testing of prototypes before production. The rights to re-engineer a product belong solely to the original owner of the design or relevant patent.

Advantages of Software Re-engineering:

1. Reduced Risk: As the software is already existing, the risk is less as compare to developing a new software
2. Reduced Cost: The cost of re-engineering is significantly less than the costs of developing a new software

Steps Involved in Software Re-Engineering:

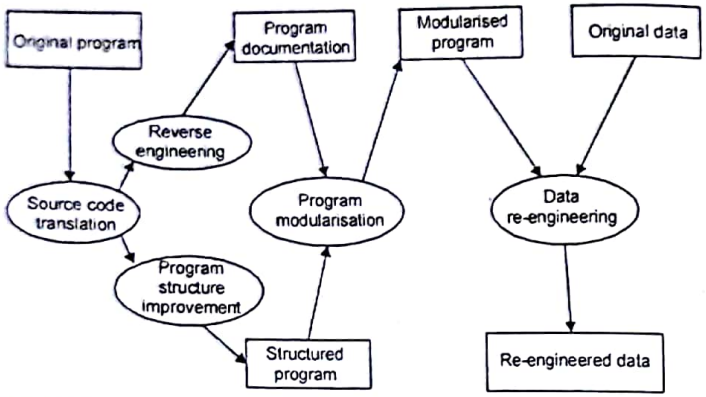
1. Source Code Translation
2. Reverse Engineering
3. Program Structure improvement
4. Program Modularisation
5. Data re-engineering

Reverse Engineering

Unlike re-engineering, reverse engineering takes a finished product with the aim of discovering how it works by testing it. Typically this is done by companies that seek to infiltrate a competitor's market or understand its new product. In doing so they can produce new products while allowing the original creator to pay all the development costs and take all the risks involved with creating a new product. Analysis of a product in this way is done without technical drawings or prior knowledge of how the device works, and the basic method used in reverse engineering begins by identifying the system's components, followed by an investigation into the relationship among these components.

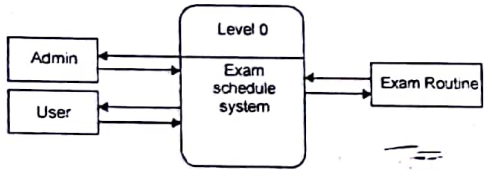
Reverse engineering is often used by companies to copy and understand parts of a competitors product, which is illegal, to find out how their own products work in the event that the original plans were lost, in order to effect repair or alter them. Reverse engineering products is illegal under the laws of many countries, however it does happen. There have been celebrated cases of reverse engineering in the third world.

Overall software re-engineering process can be shown as:



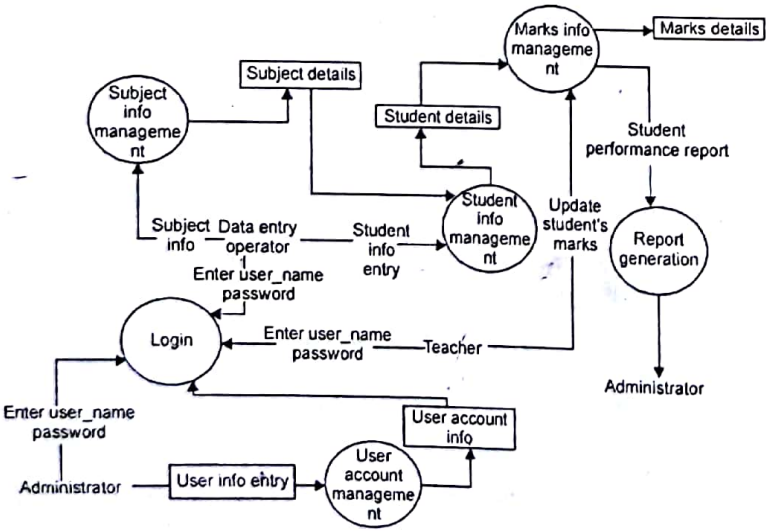
Q.7. (a) Draw 0,1 and 2 level DFD for university examination management system

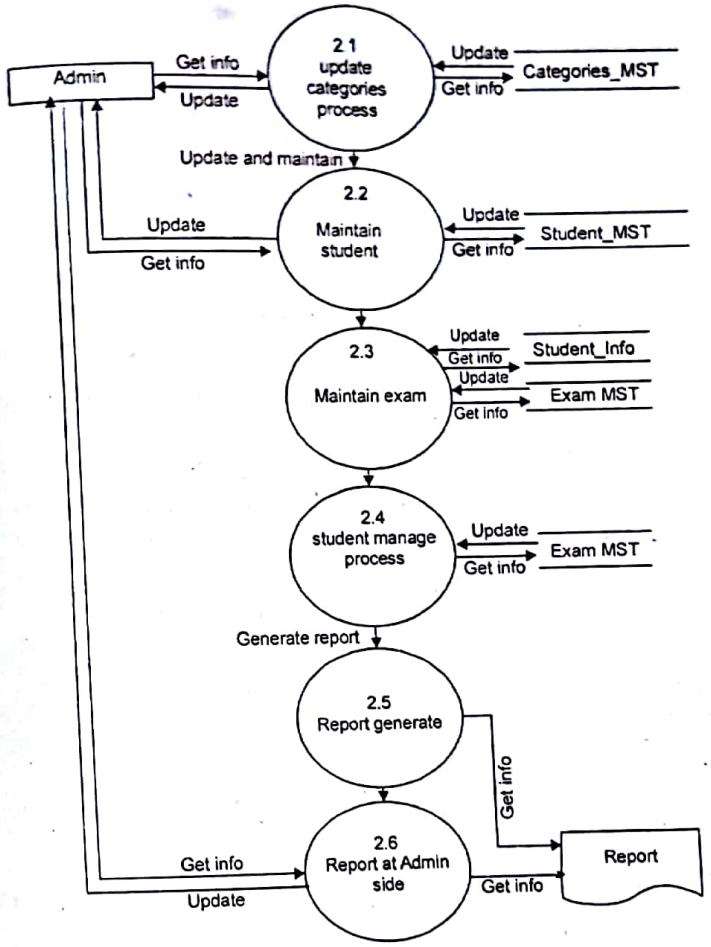
Ans. Level 0 DFD



Context Level of DFD

Level 1





Level 2

Q.7. (b) Explain Walston and Felix model. **(3)**

Ans. For any new software project, it is necessary to know how much will it cost to develop and how much development time will it take. Different models are proposed for this estimation. The model developed by Walston and Felix provides a relationship between delivered lines of source code(L in thousand of lines) and effort E(E in person-months) and is given by following equations:

$$E = 5.2L^{0.91}$$

Duration of the development (D in months) is given by:

$$D = 4.1L^{0.36}$$

Data collected on 60 software projects, representing a wide variety of applications and size shows a relationship between productivity and productivity index I.

The productivity index uses 29 variables which are found to be highly corelated to productivity as follows:

$$I = \Sigma^{29}_{i=1} W_i X_i$$

where W_i is a factor weight for ith variable and X_i={–1,0,+1}. The estimator gives X_i one of the values -1,0 or 1 depending on whether the variable decreases, has no effect or increases the productivity respectively.

Q.7. (c) Explain the prototyping model. What is the effect of designing a prototyping on the overall cost of the project? **(4.5)**

Ans. Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

he basic idea in Prototype model is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Prototype model is a software development model. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.

Following is a stepwise approach explained to design a software prototype.

• Basic Requirement Identification

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

• Developing the initial Prototype

The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed. While, the workarounds are used to give the same look and feel to the customer in the prototype developed.

• Review of the Prototype

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

• Revise and Enhance the Prototype

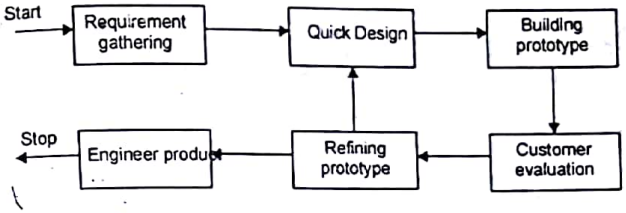
The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like – time and budget constraints and technical feasibility of the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

Prototypes can have horizontal or vertical dimensions. A Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A Vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

Advantages of Prototype model:

• Users are actively involved in the development

• Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.



Prototyping Model

• Errors can be detected much earlier.

• Quicker user feedback is available leading to better solutions.

• Missing functionality can be identified easily

• Confusing or difficult functions can be identified

Disadvantages of Prototype model:

• Leads to implementing and then repairing way of building systems.

• Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

• Incomplete application may cause application not to be used as the full system was designed

• Incomplete or inadequate problem analysis.

When to use Prototype model:

• Prototype model should be used when the desired system needs to have a lot of interaction with the end users.

• Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.

• Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Effect of Prototyping on the overall cost of project

The prototype may be a usable program, but is not suitable as the final software product. The reason may be poor performance, maintainability or overall quality. The code for the prototype is thrown away, however the experience gathered from developing the prototype helps in developing the actual system. Therefore, the development of a prototype might involve extra cost, but overall cost might turnout to be lower than that of an equivalent system developed using the waterfall model.

Q.8. Write short notes on any four of the following: **(12.5)**

Q.8. (a) QFD and FAST techniques

FAST

Facilitated Application Specification Technique ("FAST")

- It is a technique for requirements elicitation for software development.
- The objective is to close the gap between what the developers intend and what users expect.
- It is a team-oriented approach for gathering requirements.

Many different approaches to FAST have been proposed like joint application development (JDA), which was developed by IBM and the METHOD, which was developed by Inc. Falls church, VA. Each approach has some variation with the other approaches, but all follow these basic guidelines.

- A meeting is conducted at a neutral site and attended by both software engineering analysts and customers.
- Rules for preparation and participation are established.
- An agenda is suggested that is formal enough to cover all important point but informal enough to encourage the free flow of ideas.
- A facilitator (can be a customer a developer or an outsider) is assigned to control the meeting.
- A definition mechanism (can be work sheets flip charts or wall stickers or chat room) is used where how meeting will be conducted.
- The goal of this meeting is to identify the problem propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirement in an atmosphere for the achievement of the goal.

Using these guidelines as base a serried of the steps activities are performed that idea up to the meeting occur during the follow the meeting. These steps are discussed one by one:

- In the beginning a formal meeting is conducted between developer and customer, Q&A session is started to have the general idea about the problem, desired solution, customer and benefits.
- Then, customer and developer both write a separate one or two page. Product request document. This is just like a small report with brief description.
- Next, step is to set a meeting place, time and date for FAST. Parallel a facilitator and attendees of the FAST (both the development and customer user organization sides) are selected.
- After that the product request reports are distributed to all the attendees before the meeting starts.
- Additionally before the meeting each attendee is asked to make a list of three objects or instances:
- Objects that will be used by the system to perform its functions (Objects of input).
- Objects that will be produced by the system (objects of output)
- Objects that will be part of the environment that surrounds the system.

QFD

Quality function deployment (QFD) is the translation of user requirements and requests into product designs. The goal of QFD is to build a product that does exactly what the customer wants instead of delivering a product that emphasizes expertise the

builder already has. he QFD method identifies and classifies customer desires, identifies the importance of those desires, identifies engineering characteristics which may be relevant to those desires, correlates the two, allows for verification of those correlations, and then assigns objectives and priorities for the system requirements.This process can be applied at any system composition level (e.g. system, subsystem, or component) in the design of a product, and can allow for assessment of differently level abstraction systems based on the output of QFDs matrices assessed for those system levels. The output of the method is generally a matrix with customer desires on one dimension and correlated nonfunctional requirements on the other dimension.The cells of matrix table are filled with the weights assigned to the stakeholder characteristics where those characteristics are affected by the system parameters across the top of the matrix. At the bottom of the matrix, the column is summed, which allows for the system characteristics to be weighted according to the stakeholder characteristics.

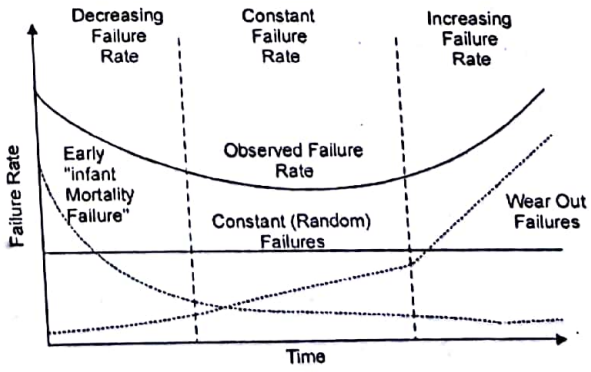
System parameters not correlated to stakeholder characteristics, may be unnecessary to the system design and are identified by empty matrix columns, while stakeholder characteristics (identified by empty rows) not correlated to system parameters indicate "characteristics not address by the design parameters". System parameters and stakeholder characteristics with weak correlations potentially indicate missing information, while matrices with "too many correlations" indicate that the stakeholder needs may need to be refined.

Q.8. (*b*) Bath tub curve of hardware reliability.

Ans. The bathtub curve is widely used in reliability engineering. It describes a particular form of the hazard function which comprises three parts:

- The first part is a decreasing failure rate, known as early failures.
- The second part is a constant failure rate, known as random failures.
- The third part is an increasing failure rate, known as wear-out failures.

In technical terms, in the early life of a product adhering to the bathtub curve, the failure rate is high but rapidly decreasing as defective products are identified and discarded, and early sources of potential failure such as handling and installation error are surmounted. In the mid-life of a product—generally, once it reaches consumers—the failure rate is low and constant. In the late life of the product, the failure rate increases, as age and wear take their toll on the product. Many consumer product life cycles strongly exhibit the bathtub curve.



Infant Mortality What Causes It and What to Do About It?

From a customer satisfaction viewpoint, infant mortalities are unacceptable. They cause "dead-on-arrival" products and undermine customer confidence. They are caused by defects designed into or built into a product. Therefore, to avoid infant mortalities, the product manufacturer must determine methods to eliminate the defects. Appropriate specifications, adequate design tolerance and sufficient component derating can help, and should always be used, but even the best design intent can fail to cover all possible interactions of components in operation. In addition to the best design approaches, stress testing should be started at the earliest development phases and used to evaluate design weaknesses and uncover specific assembly and materials problems. Tests like these are called HALT (Highly Accelerated Life Test) or HAST (Highly Accelerated Stress Test) and should be applied, with increasing stress levels as needed, until failures are precipitated. The failures should be investigated and design improvements should be made to improve product robustness. Such an approach can help to eliminate design and material defects that would otherwise show up with product failures in the field.

Q.8. (c) Various risk management activities

Risk Management

Ans. Refer Q.4. (c) of First Term Exam. 2016

Q.8. (d) Data Dictionary

A data dictionary is a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them. A first step in analyzing a system of objects with which users interact is to identify each object and its relationship to other objects. This process is called data modeling and results in a picture of object relationships. After each data object or item is given a descriptive name, its relationship is described (or it becomes part of some structure that implicitly describes relationship), the type of data (such as text or image or binary value) is described, possible predefined values are listed, and a brief textual description is provided. This collection can be organized for reference into a book called a data dictionary.

When developing programs that use the data model, a data dictionary can be consulted to understand where a data item fits in the structure, what values it may contain, and basically what the data item means in real-world terms. For example, a bank or group of banks could model the data objects involved in consumer banking. They could then provide a data dictionary for a bank's programmers. The data dictionary would describe each of the data items in its data model for consumer banking (for example, "Account holder" and "Available credit").

A data dictionary is invaluable for documentation purposes, for keeping control information on corporate data, for ensuring consistency of elements between organizational systems, and for use in developing databases. Data dictionary software packages are commercially available, often as part of a CASE package or DBMS. DD software allows for consistency checks and code generation. It is also used in DBMSs to generate reports.

Q.8. (e) Code inspection and Code walk-through

Ans. Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage. The main purpose of code inspection is to find defects and it can also spot any process improvement if any. An

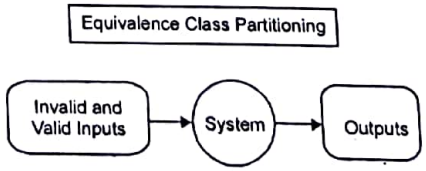
inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review. Preparation before the meeting is essential, which includes reading of any source documents to ensure consistency. Inspections are often led by a trained moderator, who is not the author of the code. The inspection process is the most formal type of review based on rules and checklists and makes use of entry and exit criteria. It usually involves peer examination of the code and each one has a defined set of roles. After the meeting, a formal follow-up process is used to ensure that corrective action is completed in a timely manner.

Code Walkthrough is a form of peer review in which a programmer leads the review process and the other team members ask questions and spot possible errors against development standards and other issues. The meeting is usually led by the author of the document under review and attended by other members of the team. Review sessions may be formal or informal. Before the walkthrough meeting, the preparation by reviewers and then a review report with a list of findings. The scribe, who is not the author, marks the minutes of meeting and note down all the defects/issues so that it can be tracked to closure. The main purpose of walkthrough is to enable learning about the content of the document under review to help team members gain an understanding of the content of the document and also to find defects.

Q.8. (f) Equivalence class testing and Integration testing

Ans. Equivalence partitioning or equivalence class partitioning (ECP) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing a software due to lesser number of test cases.

Equivalence partitioning is typically applied to the inputs of a tested component, but may be applied to the outputs in rare cases. The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object. The idea of equivalence class testing is to identify test cases by using one element from each equivalence class. If the equivalence classes are chosen wisely, the potential redundancy among test cases can be reduced.



Integration Testing

Upon completion of unit testing, the units or modules are to be integrated which gives raise to integration testing. The purpose of integration testing is to verify the 7functional, performance, and reliability between the modules that are integrated. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

Integration testing: Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. See also component integration testing, system integration testing.

Component integration testing: Testing performed to expose defects in the interfaces and interaction between integrated components.

System integration testing: Testing the integration of systems and packages; testing interfaces to external organizations (*e.g.* Electronic Data Interchange, Internet).

FIRST TERM EXAMINATION [SEPT. 2017]
FIFTH SEMESTER [B.TECH]
SOFTWARE ENGINEERING [ETCS–303]

Time : 1½ hrs. **M.M. : 30**

Note: *Q.No 1 is compulsory. Attempt any two more questions from the rest.*

Q.1. Explain in brief:

Q.1. (a) Why it is difficult to improve a software process. (2)

Ans. It is difficult to improve a software process due to following reasons:

1. Lack of knowledge- several software developers aren't aware of best practices of industry. In fact best practices obtainable in literature aren't being used widespread in software development.

2. Not enough time-There is forever a shortage of time because upper management are always demanding more software of higher quality in minimum possible time. Unrealistic schedule occasionally leave insufficient time to do the essential project work.

3. Wrong motivations-The process enhancement initiatives are taken for wrong reasons like sometimes contractor is demanding achievement of CMM or occasionally senior management is directing the organization to achieve CMM without a clear explanations why improvement was needed and its benefits.

4. Insufficient commitments-The software enhancement fails due to lack of true commitment. Management sets no outlook from the development community regarding process improvement.

Q.1. (b) What are the characteristics of good SRS? (2)

Ans. Any good requirement should have these 6 characteristics:

- Complete
- Consistent
- Feasible
- Modifiable
- Unambiguous
- Testable

Q.1. (c) What is the need to feasibility study? (2)

Ans. In a projects lifecycle, the project **feasibility** study is the second document that is created following the business case. The **purpose** of this study is to determine the factors that will make the business opportunity that was presented in the business case a success. **Feasibility studies** can be **used** in many ways but primarily focus on proposed business ventures. Farmers and others with a business idea should conduct a **feasibility study** to determine the viability of their idea before proceeding with the development of a business.

Q.1. (d) What are data structure matrices? (2)

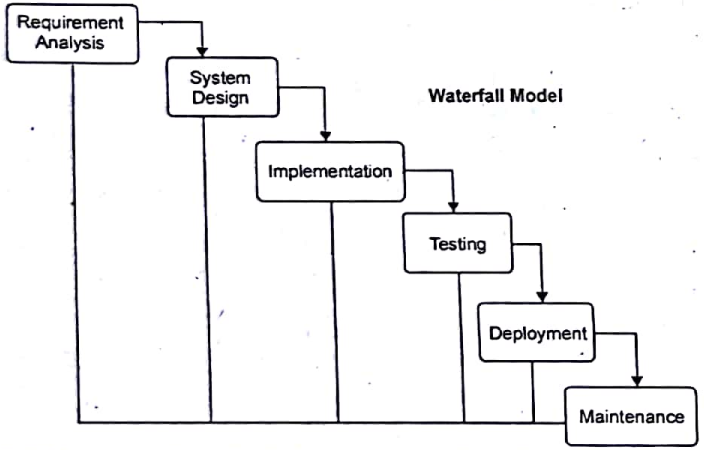
Ans. A **data structure** is a specialized format for organizing and storing **data**. General **data structure** types include the array, the file, the record, the table, the tree, and so on. Any **data structure** is designed to organize **data** to suit a specific purpose so that it can be accessed and worked with in appropriate ways. **Data structures** are building blocks of many things you want to do. If you know the uses for each **data structure**, its weaknesses and strong points then you can easily solve problems.

Q.1. (e) What is software prototyping? **(2)**

Ans. Software prototyping is the activity of creating **prototypes** of **software** applications, i.e., incomplete versions of the **software** program being developed. It is an activity that can occur in **software** development and is comparable to **prototyping** as known from other fields, such as mechanical engineering or manufacturing.

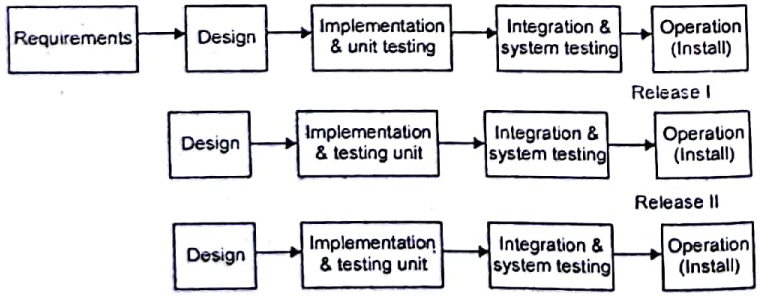
Q.2. (a) Explain generic Waterfall Model for software development? **(5)**

Ans. The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The Waterfall model is the earliest SDLC approach that was used for software development. The following illustration is a representation of the different phases of the Waterfall Model.



Q.2. (b) Compare Iterative Enhancement model with Evolutionary process model? **(5)**

Ans. Iterative Enhancement Model: This model has the similar phases as the waterfall model, but with fewer restrictions. In general the phases occur in the same order as in the waterfall model but these may be conducted in several cycles. A utilizable product is released at the end of the each cycle with each release providing additional functionality.

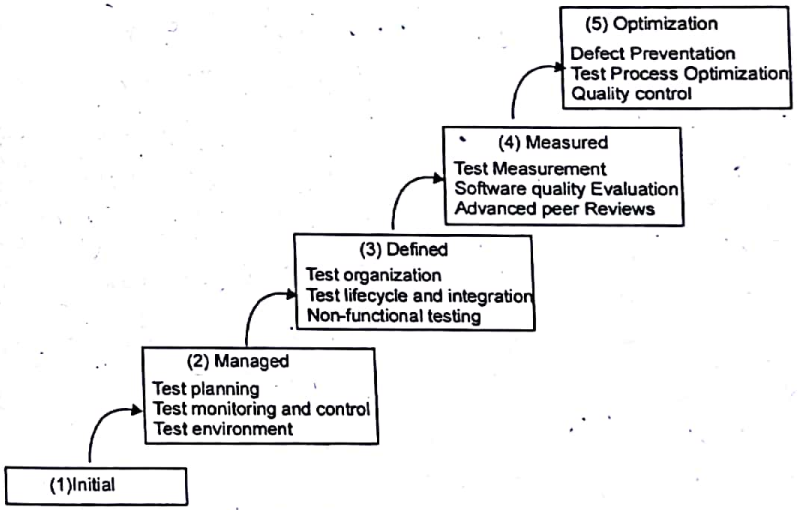


Evolutionary Development Model: Evolutionary development model bear a resemblance to iterative enhancement model. The similar phases as defined for the waterfall model occur here in a cyclical fashion. This model is different from iterative enhancement model in the sense that this doesn't require a useable product at the end of each cycle. In evolutionary development requirements are implemented by category rather than by priority.

Q.3. (a) Why CMM is used? Explain in detail the process maturity levels in SEI's CMM? **(5)**

Ans. The Software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

Levels of CMM



• **Level One : Initial** – The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.

• **Level Two: Repeatable** – This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality. The process is in place to repeat the earlier successes on projects with similar applications. Program management is a key characteristic of a level two organization.

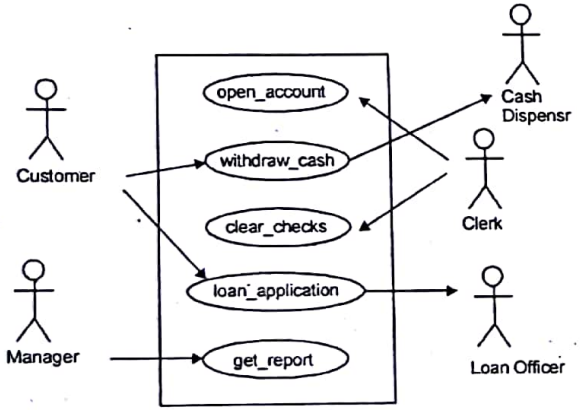
• **Level Three: Defined** – The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing,testing and maintaining the application.

• **Level Four: Managed** – Management can effectively control the software development effort using precise measurements. At this level, organization set a quantitative quality goal for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.

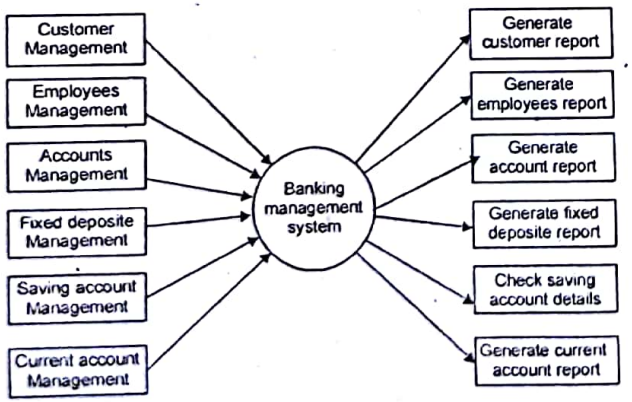
• **Level Five: Optimizing** – The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. At this level, changes to the process are to improve the process performance and at the same time maintaining statistical probability to achieve the established quantitative process-improvement objectives.

Q.3. (b) Consider the problem of Banking Managment System, design use Case Diagram Level 1 DFD and ER Diagram.
(5)

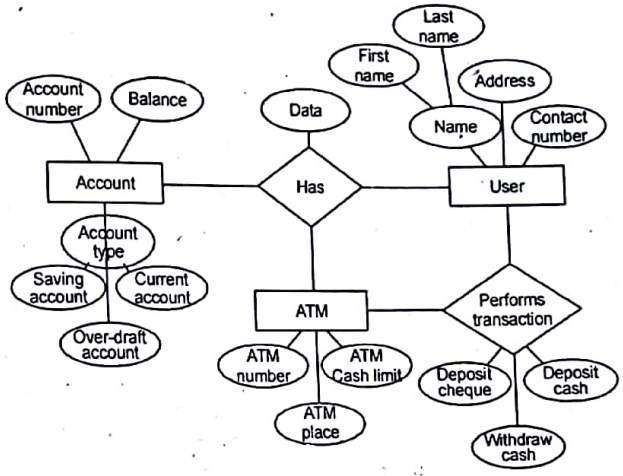
Ans.



USE CASE DIAGRAM



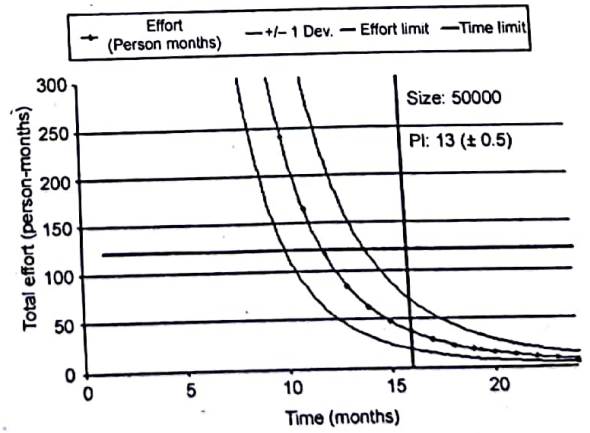
First Level DFD-Banking Management System



ER Diagram of Banking System

Q.4. Explain Putnam resource allocation model? Describe the trade-off between time verses cost in putnam resource allocation model? **(5)**

Ans. The **Putnam model** is an empirical software effort estimation model. As a group, empirical models work by collecting software project data (for example, effort and size) and fitting a curve to the data. Future effort estimates are made by providing size and calculating the associated effort using the equation which fit the original data (usually with some error). An estimated software size at project completion and organizational process productivity is used. Plotting *effort* as a function of *time* yields the *Time-Effort Curve*. The points along the curve represent the estimated total effort to complete the project at some *time*. One of the distinguishing features of the Putnam model is that total effort decreases as the time to complete the project is extended. This is normally represented in other parametric models with a schedule relaxation parameter.



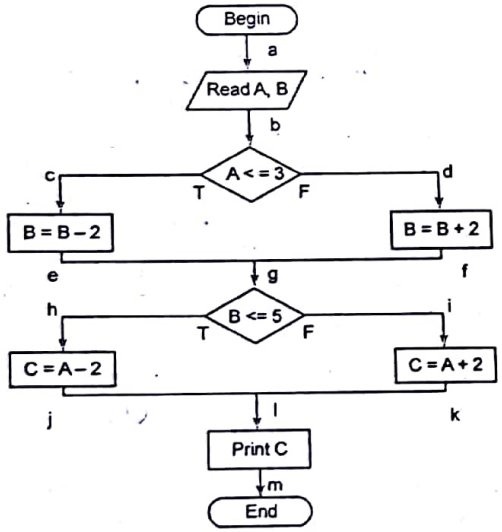
This estimating method is fairly sensitive to uncertainty in both *size* and *process productivity*. Putnam advocates obtaining process productivity by calibration.

Q.4. (b) Write a program to find the largest number among three. Also, find the Cyclomatic Complexity of the program using three different methods. (5)

Ans. The cyclomatic complexity is :

M = E – N + 2P where:

- **E = the number of edges of the graph**
- **N = the number of nodes of the graph**
- **P = the number of connected components**



Here E = 11, N = 10 and P = 1. Hence M = 10 - 11 + (2x1) = 1.

END TERM EXAMINATION [DEC. 2017]
FIFTH SEMESTER [B.TECH]
SOFTWARE ENGINEERING [ETCS–303]

Time : 3 hrs. **M.M. : 75**

***Note:** Attempt all questions as directed. Internal choice is indicated.*

Q.1. Attempt Any five questions from the following:

Q.1. (a) What are the components of a Software? Discuss how Software differ from a program? (5)

Ans. Software components of a computer system have no physical presence, they are stored in digital form within computer memory. There are different categories of software, including *system software*, *utilities* and *applications software*. **System software** is the software used to manage and control the hardware components and which allow interaction between the hardware and the other types of software. **Utility software** is software such as anti-virus software, firewalls, disk defragmenters and so on which helps to maintain and protect the computer system but does not directly interface with the hardware. **Applications software** (also known as 'apps') are designed to allow the user of the system complete a specific task or set of tasks. Software is broad term that includes Program , data structure documentation that is generated during life cycle of software development and installation files which are used to perform a computing task .

Basically if we see , Software is a term that includes the following :

1. Program 2. Installation Manual 3. Documentation

Program: It is the set of instructions executed on computer .That is Software is a Broad term and Program is a narrow term in compared to software .

Q.1. (b) What is the difference between live variable and variable span? Explain capability maturity model in brief? (5)

Ans. • Number of variable (VARS): In this metric, Number of variables used in the program are counted.

• Total number of occurrence of variable (N_2): In this metric, total number of occurrence of variables are computed

$$\text{Average no of live variable (LV)} = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statement}}$$

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.

t **Optimizing(Maturity level 5):** In this level the focus is on continually improving process performance through both incremental and innovative technological changes improvement

Q.1. (c) Define module cohesion and explain different type of cohesion? High level of cohesion and low Level of coupling is required for good quality design. Why and How? (5)

Ans. The cohesion of a module represents how tightly internal element of a module are bound with one another.

Coincidental cohesion:- This occurs when there is no relationship among elements of a module. They can occur if an existing program modularized by chopping it into pieces and making differ piece of modules i.e. it performs a set of tasks that are related to each other very loosely. The modules contain a random collection of function.

Logical cohesion:- A module having logical cohesion if there are some logical relationship between elements of a modules i.e. elements of a module performs the operation.

Temporal cohesion:- It is same as logical cohesion except that the element must be executed in same time. Set of function responsible for initialization, startup, the shutdown of the same process. It is higher than logical cohesion since all elements are executed together. This avoids the problem of passing the flag.

Procedural cohesion:- It contains that belongs to a procedural unit in which a certain sequence of steps has to be carried out in a certain order for achieving an objective.

Communicational cohesion:- A module is said to have Communicational cohesion if all function of module refers to an update the same data structure.

Sequential cohesion:- A module is said to have sequential cohesion if element module from different parts of the sequence. When the output from one element of the sequence is input to the next element of a sequence

Cohesion	Coupling
While designing you should strive for high cohesion i.e., a cohesive component/module focus on a single task (i.e., single-mindedness) with little interactive with other modules of the system. Cohesion is Intra–Module Concept.	While designing you should strive for low coupling i.e. Dependency between modules should be less, Coupling is Inter–Module- Concept.

Q.1. (d) What is the unique characteristic of Spiral process Model which are not present in other models? (5)

Ans. Authentic applications of the spiral model are driven by cycles that always display six characteristics. Boehm illustrates each with an example of a "hazardous spiral look-alike" that violates the invariant.

Define artifacts concurrently: Sequentially defining the key artifacts for a project often lowers the possibility of developing a system that meets stakeholder "win conditions" (objectives and constraints).

This invariant excludes "hazardous spiral look-alike" processes that use a sequence of incremental waterfall passes in settings where the underlying assumptions of the waterfall model do not apply. Boehm lists these assumptions as follows:

1. The requirements are known in advance of implementation.
2. The requirements have no unresolved, high-risk implications, such as risks due to cost, schedule, performance, safety, security, user interfaces, organizational impacts, etc.
3. The nature of the requirements will not change very much during development or evolution.
4. The requirements are compatible with all the key system stakeholders' expectations, including users, customer, developers, maintainers, and investors.
5. The right architecture for implementing the requirements is well understood.
6. There is enough calendar time to proceed sequentially.

In situations where these assumptions do apply, it is a project risk not to specify the requirements and proceed sequentially. The waterfall model thus becomes a risk-driven special case of the spiral model.

Q.1. (e) Distinguish Functional and Non-Functional Requirment for the Case of Banking System? (5)

Ans. Non-functional requirements are requirements that are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy. Non-functional requirements needed in this internet banking system are identified as performance requirements, safety requirements, security requirements and software quality attributes.

Functional Requirement Specification Requirements specifications add further information to the requirements definition. Natural language is often used to write requirements specifications. However, a natural language specification is not a particularly good basis for either a design or a contract between customer and system developer.

Transfer Funds: Definition: Transfer Funds allows customer to transfer funds between authorized accounts – own personal accounts. Requested transfer take place immediately or at a selected future date specified by customer

Open Payment: This function allows a customer to pay Immediate and future Payment to corporations that customer has not registered.

Pay Bills: The customer selects the Bill Payment functionality then the system displays Bill Payment Menu, and the customer selects one of four functionalities from Bill Payment menu

View Account: View Account allows to a customer to view today's up-to-the minute balance information on deposit (saving/current), credit card, etc.

Q.1. (f) Point out the shortcoming of the ISO 9001 certification as applied to the Software Industry. Explain the main features of CMM? (5)

Ans. Disadvantages of ISO 9000

There are certain disadvantages of ISO 9000 but these disadvantages can be removed by staff and management training. Some of the disadvantages are listed below.

Owners and managers donot have an adequate understanding of ISO 9000.

Most of companies have less funding available, therefore companies are finding difficulties to adopt ISO system.

ISO 9000 registration need heavy document workload.

ISO 9000 registration process require long time.

The People CMM stages the implementation of increasingly sophisticated workforce practices through these maturity levels.Organizations at the Initial Level typically exhibit four characteristics:

1. Inconsistency in performing practices
2. Displacement of responsibility
3. Ritualistic practices
4. An emotionally detached workforce

Frequent problems that keep people from performing effectively in low-maturity organizations include:

- Work overload
- Environmental distractions
- Unclear performance objectives or feedback
- Lack of relevant knowledge or skill
- Poor communication
- Low morale

Q.2. Attempt any one of the following.

Q.2. (a) Define the term "Software Engineering". Explain the major difference between Software Engineering and other traditional engineering disciplines? (4)

Ans. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software is known as software engineering.

The fundamental reason is that software is the only field of engineering that exists in a totally artificial environment. Most engineering fields are constrained by the physical world they seek to manipulate. Software engineering, however, is almost exclusively about constructs created by people and is usually not taught in terms of the limitations imposed by the physical world. Whether you design an aircraft or a software application, for both you need to:

- make designs
- define subsystems and components
- make prototypes
- specify and execute tests
- etc.

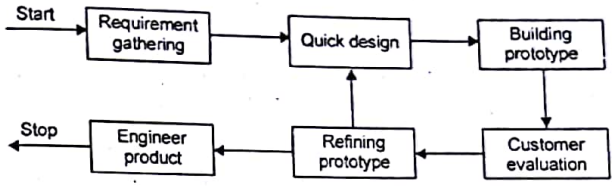
Software engineering, on the other hand, has significant factors that are an art. Yes, obviously generating the code to get the right result is a function of basic programming, and that is certainly an engineering skill, but creating the architecture that will be optimal requires a certain level of insight and taste that is beyond anything that can be found in a formula.

Q.2. (b) Discuss the Prototyping Model. What is the effect of designing a prototype on the overall cost of the project? (4)

Ans. The Prototyping Model is a systems development method (SDM) in which a prototype(an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed. This model works best in scenarios where not all of the project requirements are known in detail ahead of time. It is an iterative, trial-and-error process that takes place between the developers and the users.There are several steps in the Prototyping Model:

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the departments or aspects of the existing system.
2. A preliminary design is created for the new system.
3. A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
4. The users thoroughly evaluate the first prototype, noting its strengths and weaknesses, what needs to be added, and what should to be removed. The developer collects and analyzes the remarks from the users.

5. The first prototype is modified, based on the comments supplied by the users, and a second prototype of the new system is constructed.



Prototyping Model

To make an analogy, **prototypes** are to the final product what sketches are to wireframes/mockups. **Prototyping** is a way to "sketch with interactions" to create a rough model of usability, then refine and perfect. For this, usability testing is vital.

Q.2. (c) What are the different activities of spiral model. Why it is considered as Complex process Model? (4.5)

Ans. The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

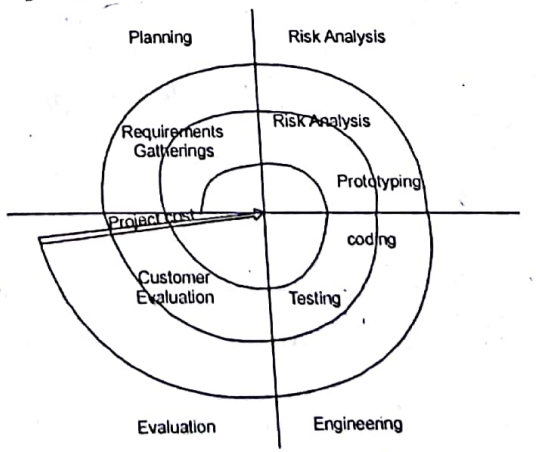
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Business Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the **risk analysis phase,** a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is **developed,** along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Diagram of Spiral model:



Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

Reasons for being considered complex:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

OR

Q.2. (a) Compare iterative enhancement model and evolutionary process model? (4)

Ans. Incremental Development

Incremental Development is a practice where the system functionalities are sliced into increments (small portions). In each increment, a vertical slice of functionality is delivered by going through all the activities of the software development process, from the requirements to the deployment.

Incremental Development (adding) is often used together with Iterative Development (redo) in software development. This is referred to as Iterative and Incremental Development (IID). The incremental model was originally developed to follow the traditional assembly line model used in factories. Unfortunately, software design and development has little in common with manufacturing physical goods. Code is the blueprint not the finished product of development. Good design choices are often 'discovered' during the development process. Locking the developers into a set of assumptions without the proper context may lead to poor designs in the best case or a complete derailing of the development in the worst.

The iterative approach is now becoming common practice because it better fits the natural path of progression in software development. Instead of investing a lot of time/effort chasing the 'perfect design' based on assumptions, the iterative approach is all about creating something that's 'good enough' to start and evolving it to fit the user's needs.

Evolutionary method: Evolutionary iterative development implies that the requirements, plan, estimates, and solution *evolve* or are refined over the course of the iterations, rather than fully defined and "frozen" in a major up-front specification effort before the development iterations begin. Evolutionary methods are consistent with the pattern of unpredictable discovery and change in new product development.

The Evolutionary model for software engineering is structured as follows:

Version 1: User Requirements Definition, System Requirements Definition, System Design/Architecture, Preliminary Design, Detailed Design, Implementation, Test and Integration, Acceptance and Certification.

Version 2: User Requirements Definition, System Requirements Definition, System Design/Architecture, Preliminary Design, Detailed Design, Implementation, Test and Integration, Acceptance and Certification.

Version 3: User Requirements Definition, System Requirements Definition, System Design/Architecture, Preliminary Design, Detailed Design, Implementation, Test and Integration, Acceptance and Certification.

Q.2. (b) What are the Components of Software? Discuss how Software differs from a program? (4)

Ans. Refer Q. 1. (a) of End Term Examination 2017.

Q.2. (c) Explain different types of empirical cost estimation models? (4.5)

Ans. Empirical estimation technique are based on the data taken from the previous project and some based on guesses and assumptions.

There are many empirical estimation technique but most popular are

(i) Expert Judgement Technique

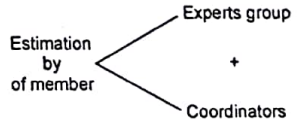
(ii) Delphi Cost Technique

Expert judgement technique: An expert makes an educated guess of the problem size after analyzing the problem thoroughly. Expert estimate the cost of different components that is modules and sub modules of the system.

Disadvantages: Human error, considering not all factors and aspects of the project, individual bias, more chances of failure.

Estimation by group of experts minimises factors such as individual oversight, lack of familiarity with a particular aspect of a project, personal bias and desired to win a contract through overly optimistic estimates.

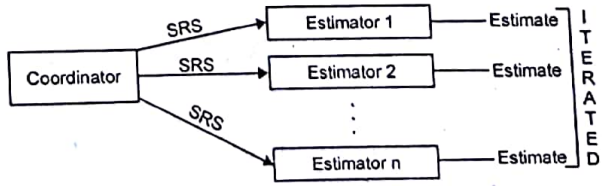
Delphi cost estimation:



Role of Members: Coordinator provide a copy of Software Requirement Specification(SRS) document and a form of recording it cost estimate to each estimator.

Estimator : It complete their individual estimate anomalously and submit to the coordinator with mentioning, if any, unusual characteristics of product which has influenced his estimation.

The coordinator and distribute the summary of the response to all estimator and they re-estimate them.



This process is iterated for several rounds

No discussion is allowed among the estimator during the entire estimation process because there may be many estimators get easily influenced by rationale of an estimator who may be more experienced or senior.

Q.3. Attempt any one of the following.

Q.3. (a) Compute the function point value for a project with the following information domain-characteristics. (6.5)

Number of user inputs = 24
Number of user outputs = 65
Number of user enquires = 12
Number of files = 23
Number of External interfaces = 7
Assumes that all Complexity adjustment values are moderate. Explain
Ans.
Function point = FP = UPF x VAF
UFP = Sum of all the complexities i.e. the 5 parameters provided in the question,
VAF = Value added Factor i.e. 0.65 + (0.01 * TDI),
TDI = Total Degree of Influence of the 14 General System Characteristics.

Table 1: Function point complexity weights.

Measurement parameter	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

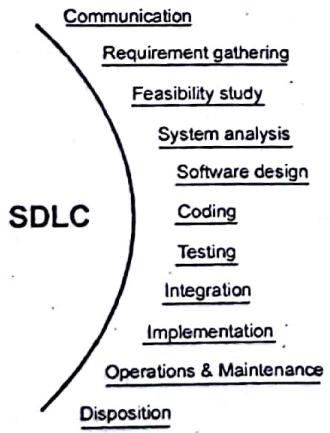
Thus function points can be calculated as:
= (24*4 + 65*5 + 12*4 + 23*10 + 28) x (0.65 + (.01 x (14 x 3))
= 727x (0.65 + 0.42)
= 727x (1.07) = 777.89

Q.3. (b) Explain the SDLC model in software development process. Describe waterfall model. (6)

Ans. Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:



Communication: This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

Requirement Gathering: This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- collecting answers from the questionnaires.

Feasibility Study: After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up.

System Analysis: At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc.

Software Design: Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design.

Coding: This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing: An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end.

Integration: Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation: This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance: This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational.

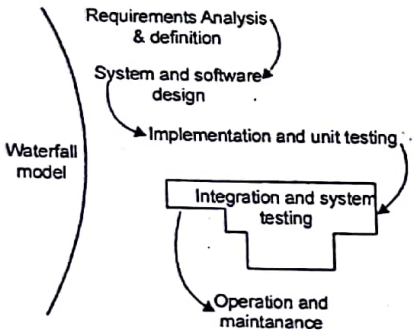
Waterfall model is a simplest model of software development paradigm.

- All the phases of sdlc will function one after in linear manner. That is, when the first phase is finished then only the second phase will start and so on.

The Waterfall model was first process model to be introduce and also called "Linear-Sequential Life Cycle Model.This type of model is basically used for the project which is small and there are "No Uncertain Requirements".

At the end of a phase, a review take place to determine if the project is on the right path and whether or not to continue or discard the project.

In this testing starts only after the development is completed and phases do not overlap.



OR

Q.3. (a) Explain COCOMO model for cost estimation. What is the limitation of COCOMO models? **(6.5)**

Ans. COCOMO model was given by Boehm in 1981. It is a hierarchy of software cost estimation models which includes basic, intermediate and detailed sub-models.

BASIC MODEL:-

This model aims at estimating most of the small to medium sized software projects in a quick and rough fashion.

It has 3 modes of software development:-

(a) ORGANIC MODE

(b) SEMIDETACHED MODE

(c) EMBEDDED MODE

(a) ORGANIC MODE

The size of project is 2 to 50 KLOC. A small team of experienced developers develops software in a very familiar environment. The deadline of project is not tight.

(b) SEMIDETACHED MODE

The size of project is 50 to 300 KLOC. A team having average previous experience on similar projects develops software in a medium environment.

The deadline of project is medium.

(c) EMBEDDED MODE

The size of project is over 300 KLOC. Real time systems, complex interfaces and a team of very little previous experienced developers are involved.

The deadline of project is tight.

INTERMEDIATE MODEL:-

The basic model allowed for a quick and rough estimate but the result will lack accuracy. So, Boehm introduced an additional set of 15 predictors called cost drivers in the intermediate model to take account of the software development environment. These cost drivers are used to adjust the nominal cost of a project to the actual project environment. Hence, the accuracy of the estimate will be increased. The cost drivers are grouped into 4 categories:-

1. Product attributes

(a) Required Software Reliability(RELY)

(b) Database size(DATA)

(c) Product Complexity(CPLX)

2. Computer attributes

(a) Execution Time Constraint(TIME)

(b) Main Storage Constraint(STOR)

(c) Virtual Machine Volatility(VIRT)

(d) Computer Turnaround Time(TURN)

3. Personal attributes

(a) Analyst Capability(ACAP)

(b) Application Experience(AEXP)

(c) Programmer Capability(PCAP)

(d) Virtual Machine Experience(VEXP)

(e) Programming Language Experience(LEXP)

4. Project attributes

(a) Modern Programming Practices(MODP)

(b) Use Of Software Tools(TOOL)

(c) Required Development Schedule(SCED)

DETAILED MODEL:

It offers a means for processing all the project characteristics to construct a software estimate. The detailed model introduces two more capabilities:-

(1) Phase-sensitive effort multipliers

Some phases are more affected than others by factors defined by the cost drivers. The detailed model provides a set of phase sensitive effort multipliers for each cost driver. This helps in determining the manpower allocation for each phase of the project.

(2) Three-level product hierarchy

Three product levels are defined. These are module, subsystem and system levels.

Limitation of COCOMO Models:

a. COCOMO model ignores requirements and all documentation.

b. It ignores customer skills, cooperation, knowledge and other parameters.

c. It oversimplifies the impact of safety/security aspects.

d. It ignores hardware issues

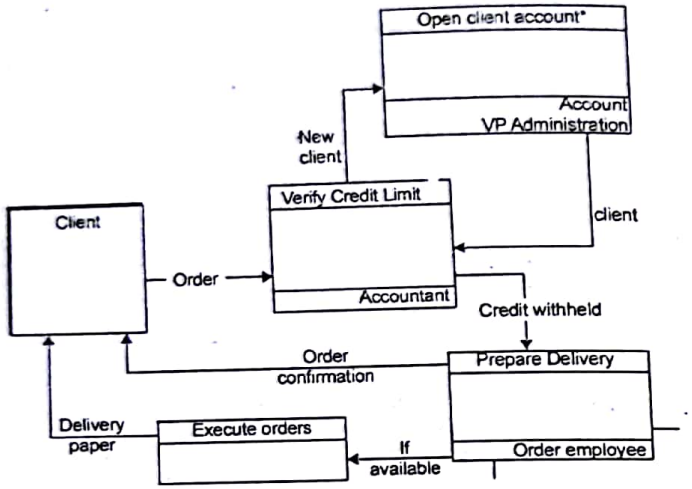
e. It ignores personnel turnover levels

f. It is dependent on the amount of time spent in each phase.

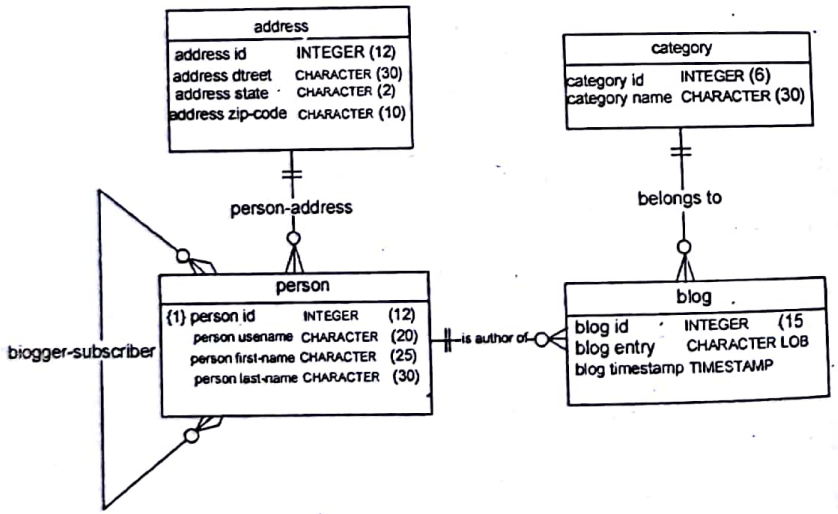
Q.3. (b) Distinguish between ER diagram and DFD. Draw a DFD of any suitable example as per your choice? **(6)**

Ans. A **DFD** (Data Flow Diagram) is a graphical representation of the **data in motion** within a system (e.g. a business, an organization, an application) that is subject to analysis. On the highest (least detailed) level, it identifies all flows from system-external sources and to system-external destinations. With further decomposition of the system, the lower level diagrams show how data elements (that compose a data flow) are processed and passed on to subsequent processes or a store.

An **ERD** (Entity Relationship Diagram) is a graphical representation of the **data in rest** within a system (e.g. a business, an organization, an application) that is subject to analysis. On the conceptual (business) level, it shows how the data elements are logically grouped in entities (i.e. they describe e.g. a person, an address, a blog article) and the relationships between these entities



(e.g. a person is author of certain blog articles), on the physical level it displays how data elements are technically stored e.g. in a database.



Since DFDs and ERDs are obviously more than just "drawings", they ought to be created and maintained using business process and data modeling tools, i.e. software that supports the methods of decomposition and description as well as the method-related rules of consistency and integrity. High-quality modeling tools will also foster the exchange of work items between DFDs and ERDs, e.g. to enforce naming standards (avoiding homonyms and inadvertent synonyms).

Q.4. Describe the role and use of coupling and cohesion in software design process through an example. Enumerate different types of coupling and cohesion. **(12.5)**

Ans. Cohesion and **Coupling deal** with the quality of an Object Oriented(OO) design. Generally, good OO design should be loosely coupled and highly cohesive. Lot of the design principles, design patterns which have been created are based on the idea of "Loose coupling and high cohesion".

The aim of the design should be to make the application:

- easier to develop
- easier to maintain
- easier to add new features
- Less Fragile.

Coupling: Coupling is the degree to which one class knows about another class. Let us consider two classes class A and class B. If class A knows class B through its interface only i.e. it interacts with class B through its API then class A and class B are said to be loosely coupled.

If on the other hand class A apart from interacting class B by means of its interface also interacts through the non-interface stuff of class B then they are said to be tightly coupled. Suppose the developer changes the class B's non-interface part i.e. non API stuff then in case of loose coupling class A does not breakdown but tight coupling causes the class A to break.

So it's always a good OO design principle to use loose coupling between the classes i.e. all interactions between the objects in OO system should use the APIs. An aspect of good class and API design is that classes should be well encapsulated.

```
// Tightly coupled class design - Bad thing
class DoTaxes {
  float rate;
  float doColorado() {
    SalesTaxRates str = new SalesTaxRates();
    rate = str.salesRate; // ouch
    // this should be a method call:
    // rate = str.getSalesRate("CO");
    // do stuff with rate
  }
}
class SalesTaxRates {
  public float salesRate; // should be private
  public float adjustedSalesRate; // should be private
  public float getSalesRate(String region) {
    salesRate = new DoTaxes().doColorado(); // ouch again!
    // do region-based calculations
    return adjustedSalesRate;
  }
}
```

Ideally, all interactions between objects in an OO system should use the APIs. In other words, the contracts, of the object's respective classes. Theoretically, if all of the classes in an application have well-designed APIs, then it should be possible for all interclass interactions to use those APIs exclusively.

Cohesion: Cohesion is used to indicate the degree to which a class has a single, well-focused purpose. Coupling is all about how classes interact with each other, on the other hand cohesion focuses on how single class is designed. Higher the cohesiveness of the class, better is the OO design.

Benefits of Higher Cohesion:

- Highly cohesive classes are much easier to maintain and less frequently changed.
- Such classes are more usable than others as they are designed with a well-focused purpose.
- Single Responsibility principle aims at creating highly cohesive classes.

```
// Less cohesive class design
class BudgetReport {
  void connectToRDBMS() {
  }
  void generateBudgetReport() {
  }
  void saveToFile() {
  }
  void print() {
  }
}
More cohesive class design
// More cohesive class design
class BudgetReport {
  Options getReportingOptions() {
  }
  void generateBudgetReport(Options o) {
  }
}
class ConnectToRDBMS {
  DBconnection getRDBMS() {
  }
}
class PrintStuff {
  PrintOptions getPrintOptions() {
  }
}
class FileSaver {
  SaveOptions getFileSaveOptions() {
  }
}
```

This design is much more cohesive because Instead of one class that does everything, we have broken the system into four main classes. Each with a very specific, or cohesive, role. Because we have built these specialized, reusable classes

TYPES OF COUPLING:

Content coupling (high)

Content coupling is said to occur when one module uses the code of other module, for instance a branch. This violates information hiding - a basic design concept.

Common coupling: Common coupling is said to occur when several module have access to same global data. But it can lead to uncontrolled error propagation and unforeseen side-effects when changes are made.

External coupling: External coupling occurs when two modules share an externally imposed data format, communication protocol, or device interface. This is basically related to the communication to external tools and devices.

Control coupling: Control coupling is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).

Stamp coupling (data-structured coupling)

Stamp coupling occurs when modules share a composite data structure and use only parts of it, possibly different parts (e.g., passing a whole record to a function that needs only one field of it).

In this situation, a modification in a field that a module does not need may lead to changing the way the module reads the record.

Data coupling: Data coupling occurs when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).

TYPES OF COHESION:

Logical cohesion: Logical cohesion is when parts of a module are grouped because they are logically categorized to do the same thing even though they are different by nature (e.g. grouping all mouse and keyboard input handling routines).

Temporal cohesion: Temporal cohesion is when parts of a module are grouped by when they are processed - the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

Procedural cohesion: Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

Communicational/informational cohesion

Communicational cohesion is when parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information).

Sequential cohesion: Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data).

Functional cohesion (best)

Functional cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module (e.g. Lexical analysis of an XML string)

Q.5. (a) What is the purpose of integration testing? How it is done? Differentiate between integration testing and system testing? (4)

Ans. INTEGRATION TESTING is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing. Tasks

- Integration Test Plan
- Prepare
- Review
- Rework
- Baseline
- Integration Test Cases/Scripts
- Prepare

- Review
- Rework
- Baseline
- Integration Test
- Perform

When is Integration Testing performed?

Integration Testing is the second level of testing performed after Unit Testing and before System Testing.

Difference between System Testing and Integration Testing:

System Testing	Integration Testing
1. Testing the completed product to check if it meets the specification requirements.	1. Testing the collection and interface modules to check whether they give the expected result
2. Both functional and non-functional testing are covered like sanity, usability, performance, stress an load .	2. Only Functional testing is performed to check whether the two modules when combined give correct outcome.
3. It is a high level testing performed after integration testing	3. It is a low level testing performed after unit testing
4. It is a black box testing technique so no knowledge of internal structure or code is required	4. It is both black box and white box testing approach so it requires the knowledge of the two modules and the interface
5. It is performed by test engineers only	5. Integration testing is performed by developers as well test engineers
6. Here the testing is performed on the system as a whole including all the external interfaces, so any defect found in it is regarded as defect of whole system	6. Here the testing is performed on interface between individual module thus any defect found is only for individual modules and not the entire system
7 In System Testing the test cases are developed to simulate real life scenarios	7. Here the test cases are developed to simulate the interaction between the two module

Q.5. (b) Compare Path Testing, Data flow testing and mutation testing? Why do we require these types of testing? (4)

Ans. 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. Data flow 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.

Mutation testing is a structural testing technique, which uses the structure of the code to guide the testing process. On a very high level, it is the process of rewriting the source code in small ways in order to remove the redundancies in the source code

These ambiguities might cause failures in the software if not fixed and can easily pass through testing phase undetected.

Mutation Testing Benefits:

Following benefits are experienced, if mutation testing is adopted:

- It brings a whole new kind of errors to the developer's attention.
- It is the most powerful method to detect hidden defects, which might be impossible to identify using the conventional testing techniques.
- Tools such as Insure++ help us to find defects in the code using the state-of-the-art.
- Increased customer satisfaction index as the product would be less buggy.
- Debugging and Maintaining the product would be more easier than ever.

Mutation Testing Types:

- **Value Mutations:** An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.
- **Decision Mutations:** The decisions/conditions are changed to check for the design errors. Typically, one changes the arithmetic operators to locate the defects and also we can consider mutating all relational operators and logical operators (AND, OR , NOT)
- **Statement Mutations:** Changes done to the statements by deleting or duplicating the line which might arise when a developer is copy pasting the code from somewhere else.

Path testing is an approach to testing where you ensure that every path through a program has been executed at least once. You normally use a dynamic analyzer tool or test coverage analyser to check that all of the code in a program has been executed. However, testing all paths does not mean that you will find all bugs in a program. Many bugs arise because programmers have forgotten to include some processing in their code, so there are no paths to execute. Some bugs are also related to the order in which code segments are executed. For example, if segments a, b and c are executed <a, b, c>, then the program may work properly. However, if they are executed <a, c, b> then an error may arise. It is practically impossible to test all orders of processing as well as all program paths.

Q.5. (c) Explain the Concept of Maintenance. Describe the various categories of Maintenance? Explain any software maintenance model. (4.5)

Ans. Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

• **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

• **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Maintenance Activities:

• **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages.Here, the maintenance type is classified also.

• **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.

• **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.

• **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.

• **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

• **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

• **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

• **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

Taute Maintenance Model is a type of Software Maintenance Model, which is straightforward, practical and easy to understand. It is used by developers for updating and performing modification in the software after its execution in the system. Phases of Taute Maintenance Model:The complete model can be implemented by following the flowchart diagram that illustrates the whole process from start to end by utilizing and defining each module effectively.