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

$$m_0 = \frac{k}{td\sqrt{e}} = \frac{300 \times 10 \times 100}{2.5 \times 1.648}$$

$$= \frac{12000 \times 10}{1648} = 72.82$$

(*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 faciitator

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)

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

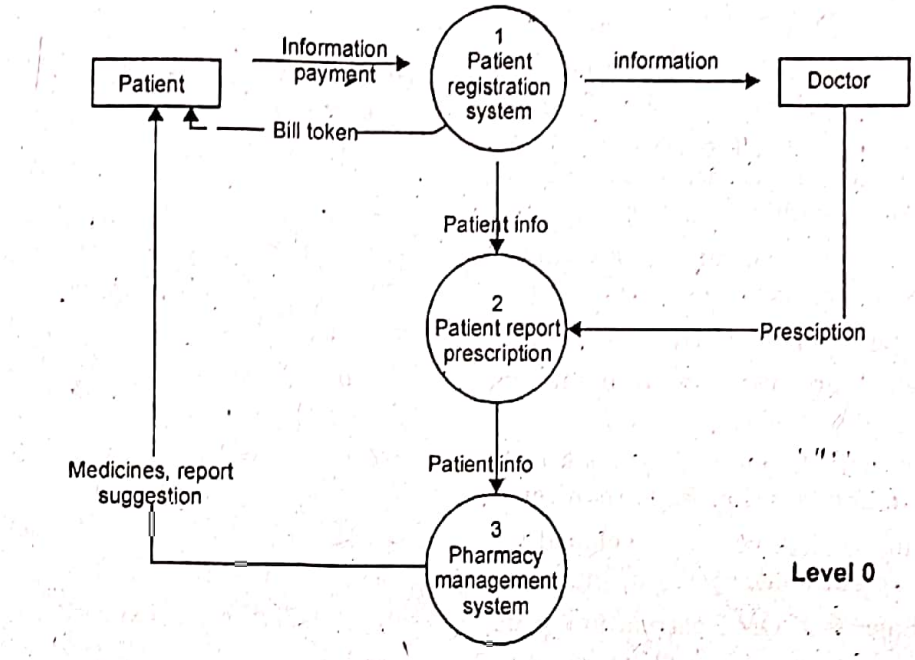
= 26 hours

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

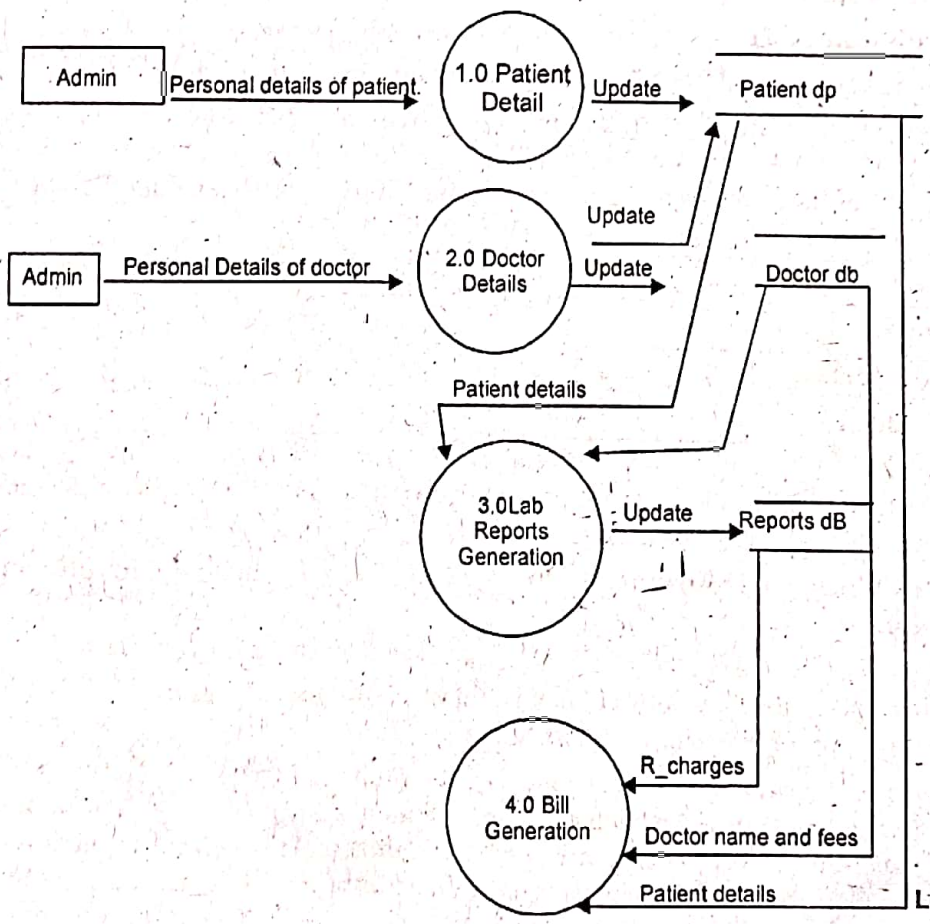
Ans. A context diagram is a top level (also known as Level 0) data flow diagram. It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities. In level 0 dfd, system is shown as one process.

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:

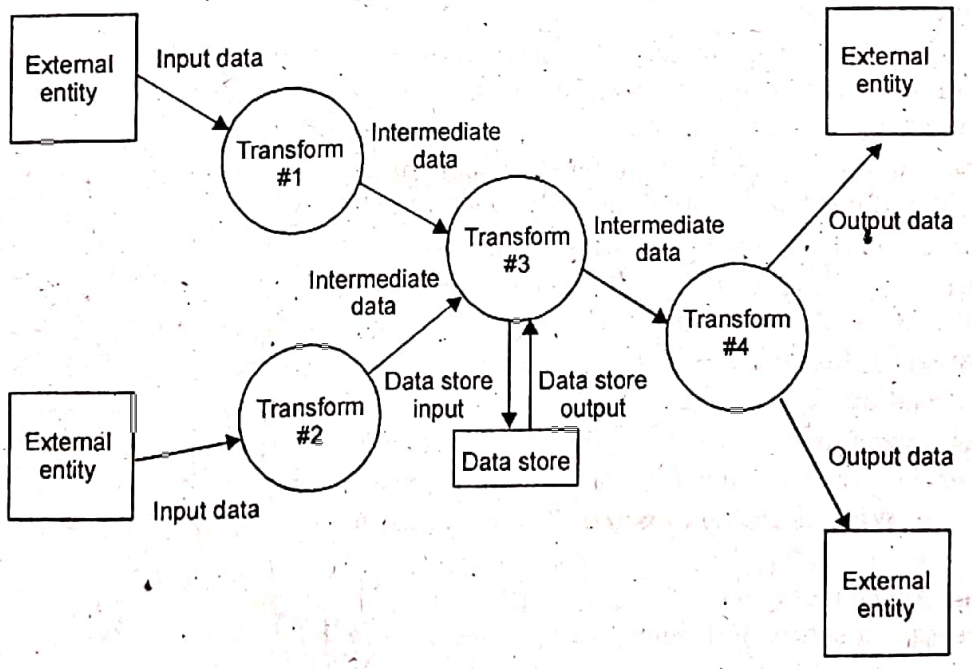
IC1 = LLOC * IFIO

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.

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)

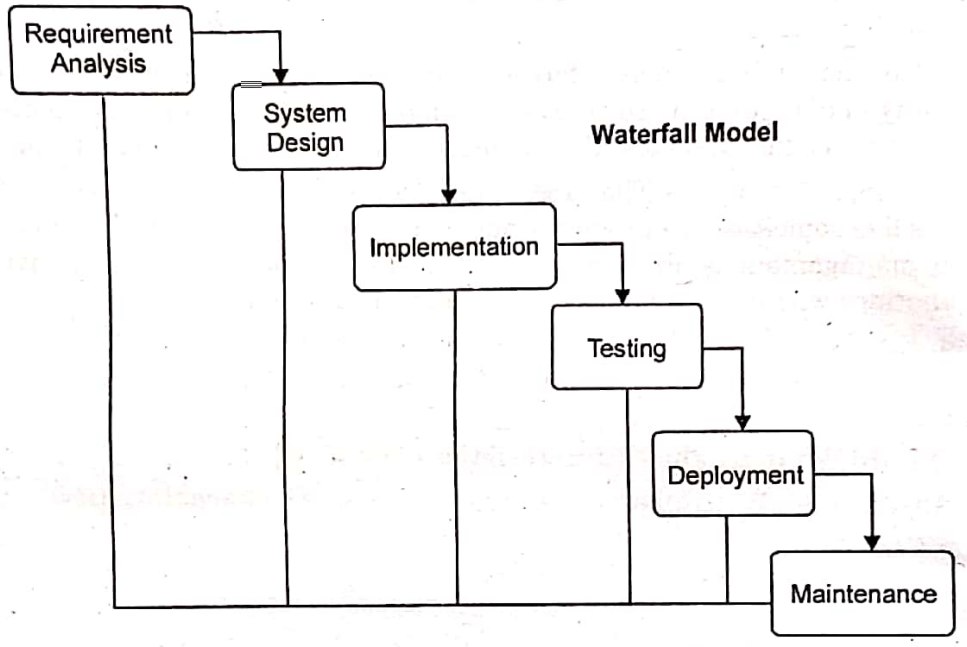
Ans. A **data structure** is a specialized format for organizing and storing **data.** General **data structure** types include the array, the file, the record, the table, the tree, and so on. Any **data structure** is designed to organize **data** to suit a specific purpose so that it can be accessed 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. (c) What is software prototyping?

Ans. Software prototyping is the activity of creating **prototypes** of **software** applications, i.e., incomplete versions of the **software** program being developed. It is a activity that can occur in **software** development and is comparable to **prototyping** known from other fields, such as mechanical engineering or manufacturing.

Q.2. (a) Explain generic Waterfall Model for software development?

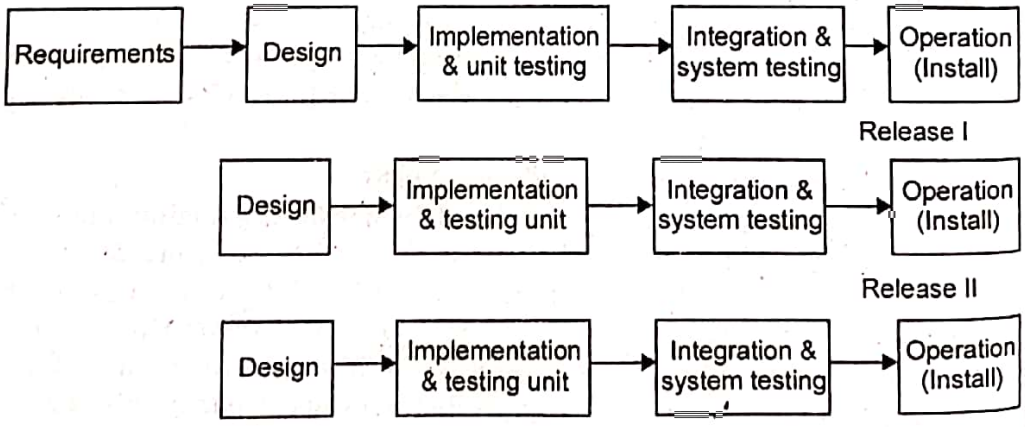
Ans. The Waterfall Model was the first Process Model to be introduced. It is al referred to as a **linear-sequential life cycle model**. It is very simple to understand an use. In a waterfall model, each phase must be completed before the next phase can beg and there is no overlapping in the phases. The Waterfall model is the earliest SDL approach that was used for software development. The following illustration is representation of the different phases of the Waterfall Model.



Waterfall Model

Q.2. (b) Compare Iterative Enhancement model with Evolutionary proce model?

Ans. Iterative Enhancement Model: This model has the similar phases as t waterfall model, but with fewer restrictions. In general the phases occur in the sa order as in the waterfall model but these may be conducted in several cycles. A utilizab product is released at the end of the each cycle with each release providing additio functionality.

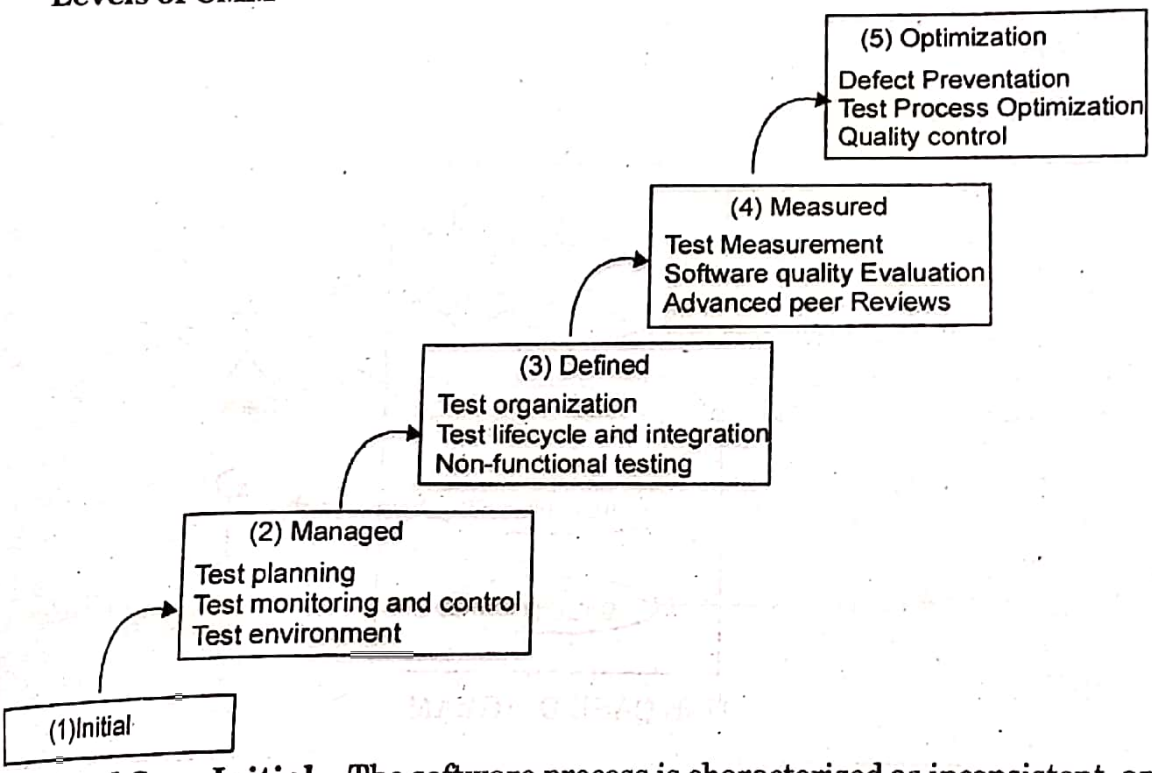


Evolutionary Development Model: Evolutionary development model bear a resemblance to iterative enhancement model. The similar phases as defined for the waterfall model occur here in a cyclical fashion. This model is different from iterative enhancement model in the sense that this doesn't require a useable 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.

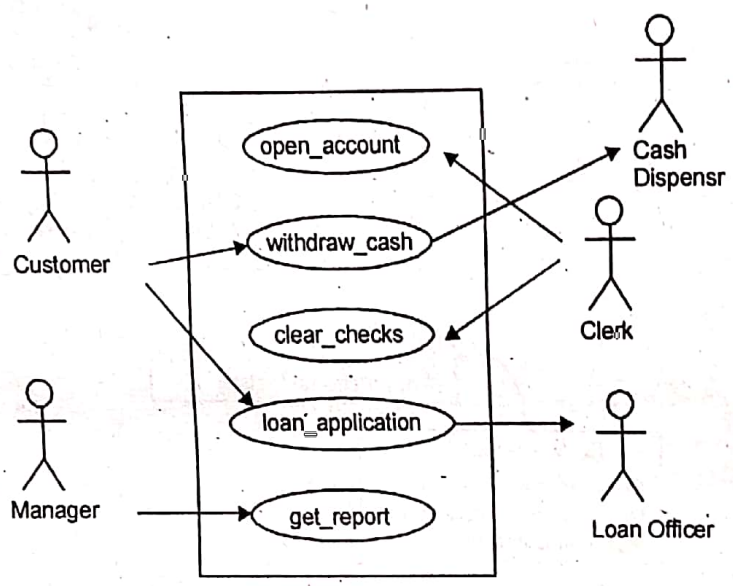
- **Level Three: Defined** – The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing,testing and maintaining the application.

• **Level Four: Managed** – Management can effectively control the softwa development effort using precise measurements. At this level, organization set quantitative quality goal for both software process and software maintenance. At th maturity level, the performance of processes is controlled using statistical and oth quantitative techniques, and is quantitatively predictable.

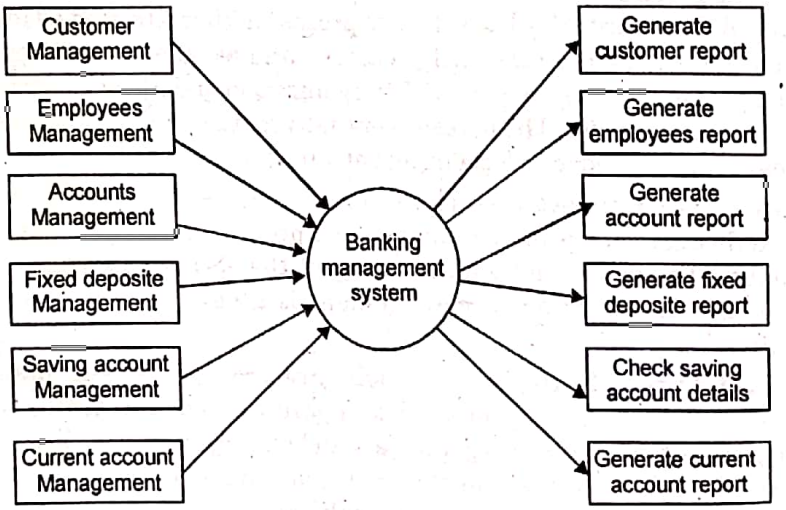
• **Level Five: Optimizing** – The Key characteristic of this level is focusing c continually improving process performance through both incremental and innovati technological improvements. At this level, changes to the process are to improve th process performance and at the same time maintaining statistical probability to achie the established quantitative process-improvement objectives.

Q.3. (b) Consider the problem of Banking Managment System, design us Case Diagram Level 1 DFD and ER Diagram. **(5)**

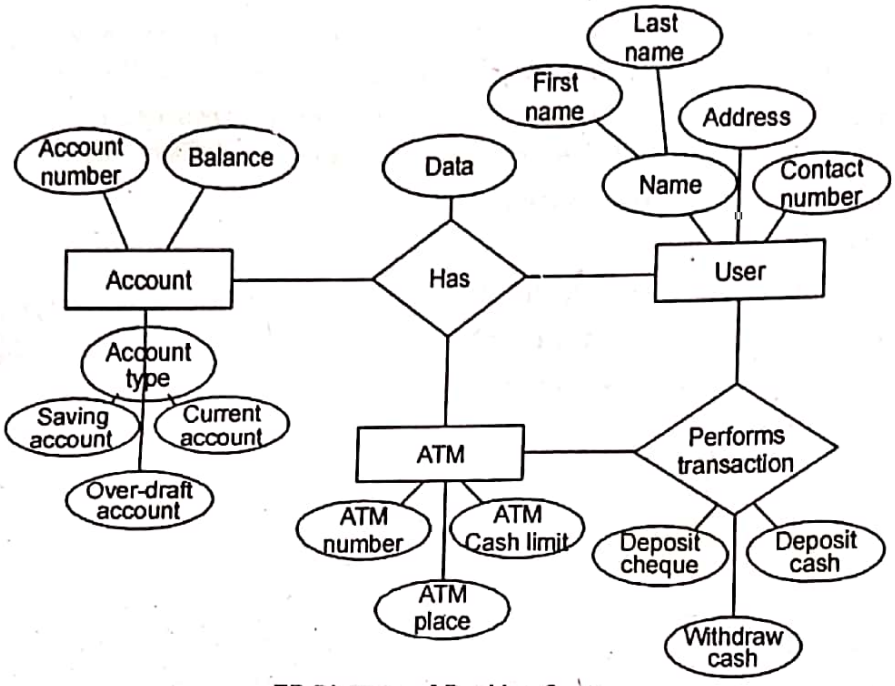
Ans.



USE CASE DIAGRAM



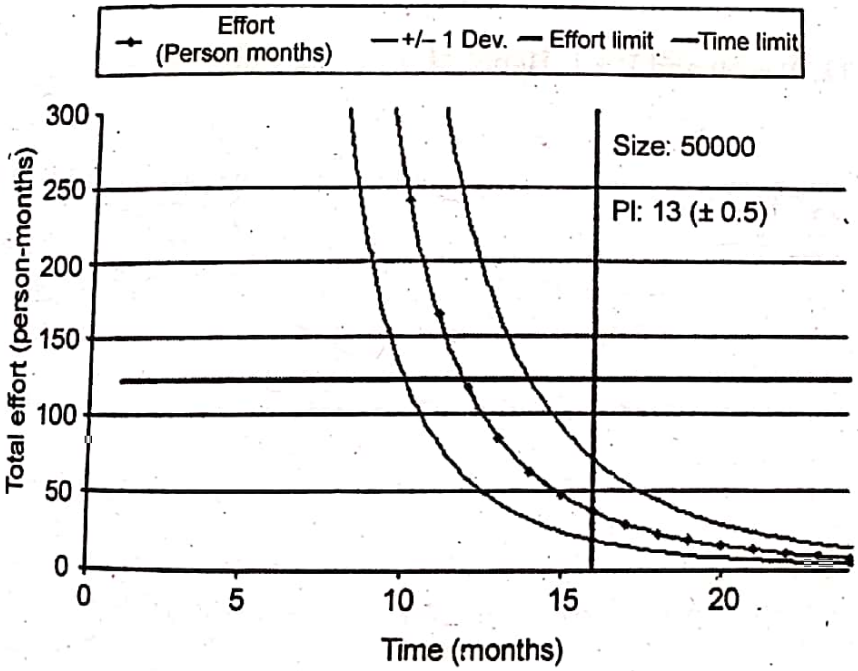
First Level DFD-Banking Management System



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.



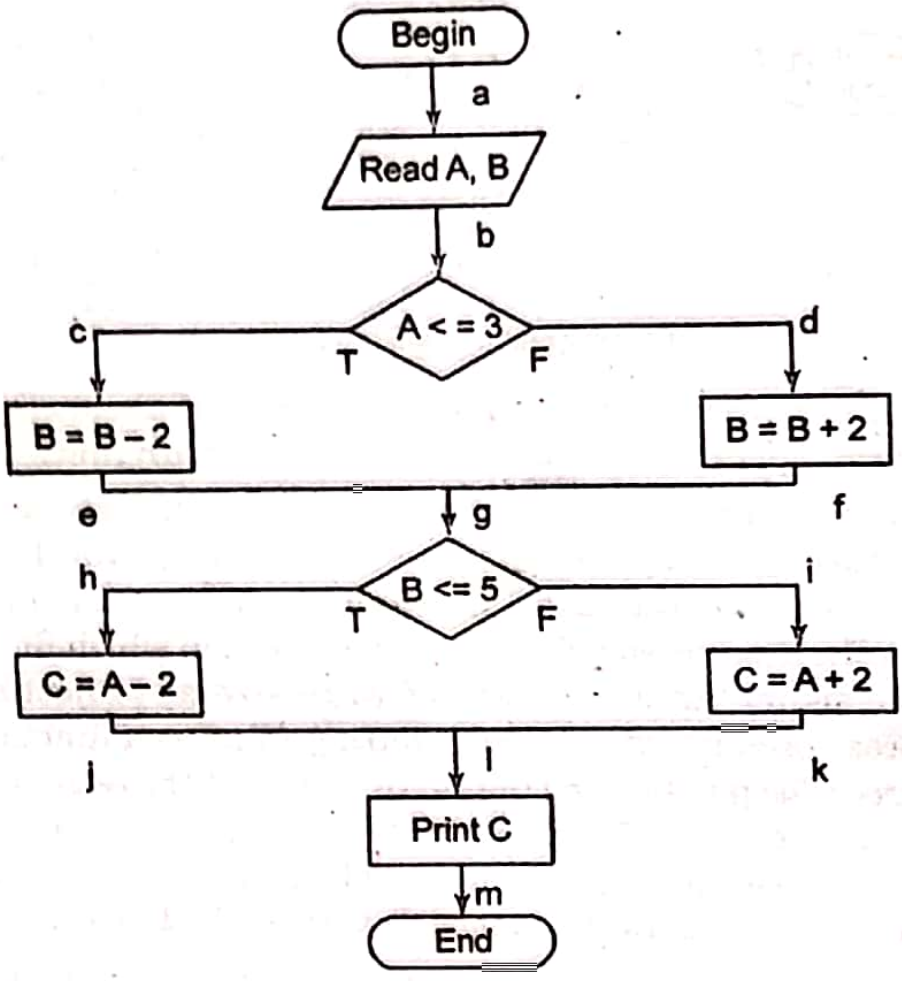
This estimating method is fairly sensitive to uncertainty in both *size* and *process productivity*. Putnam advocates obtaining process productivity by calibration.

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.

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.

FIRST TERM EXAMINATION [SEPT-2018]
FIFTH SEMESTER [B.TECH]
SOFTWARE ENGINEERING[ETCS–303]

Time : 1.5 hrs. **M.M. : 30**

Note: Q. No. 1 is compulsory. Attempt any two more questions from the rest.

Q.1. (a) Discuss the need and importance of SRS. **(2.5)**

Ans. The need and importance of SRS are:

- The users and the client get a brief idea about the software while in the initial stages.
- The purposes and the intentions as well as the expected results are properly defined. It hence lays the outline for software design.
- The desired goals are defined thereby easing off the efforts of the developers in terms of time and cost.
- It forms a basis for the agreement between the client and the developer.
- It becomes easier while transferring and using the solution elsewhere or with new customers as the basis of functioning of the software is mentioned.
- It acts as a material for reference at a later stage.
- It acts as the basis for reviews.

Q.1.(b) What are live variables? Explain. **(2.5)**

Ans. The set of variables that are assigned a value in s (in many books, KILL (s) is also defined as the set of variables assigned a value in s before any use, but this does not change the solution of the dataflow equation): The in-state of a block is the set of variables that are live at the start of the block.

– For each Variable x where is the last program point p where the a specific value of x is used.

– In other words, for x and program point p determine if the value of x at p can still be used along some path starting at p.

- If so, x is live at p
- If not x is dead at p.

Q.1.(c) What is CMM? How is it different from ISO 9001? **(2.5)**

Ans. CMM: 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 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.

Why CMM is different from ISO 9001:

CMM is specially developed for software industry.

CMM focuses on the software Engineering activities.

CMM gets into technical aspect of software engineering.

CMM has 5 levels: Initial, Repeatable ,Defined , Managed, Optimization

Q.1.(d) Compare facilitated application specification technique (FAST) with brain-storming session.

Ans. Facilitated Application Specification Technique(FAST): (2.5)

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.

Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally a document is prepared which consists of the list of requirements and their priority if possible.

Q.2.(a) Comapre the function point value for a project with the following information domain characterstics:

Number of Inputs=24

Number of User outputs=65

Number of User enquiries=12

Number of files=12

Number of external interfaces=4

Assume that all weighting factors are average. In addition to that, system has significant data communication and designed code may be moderately reusable. Assume other complexity adujustment factors as moderate. (5)

Ans.

Measurement Parameter	*Count*		*Weighing factor*		
			Simple	*Average*	*Complex*
1. Number of external inputs (EI)	24	*	4	=	96
2. Number of external outputs (EO)	65	*	5	=	325
3. Number of external inquiries (EQ)	12	*	4	=	48
4. Number of internal files (ILF)	12	*	10	=	120
5. Number of external interfaces (EIF)	4	*	7	=	28
Count-total →					617

Now F_i for moderate case = 2

So sum of all F_i ($i \leftarrow 1$ to 14) = $14 \times 2 = 28$

$$FP = \text{Count-total} \times [0.65 + 0.01 \times \Sigma (F_1)]$$
$$= 617 \times [0.65 + 0.01 \times 28]$$

$$= 617 \times [0.65 + 0.28]$$
$$= 617 \times 1.23 = 758.91 = 759$$

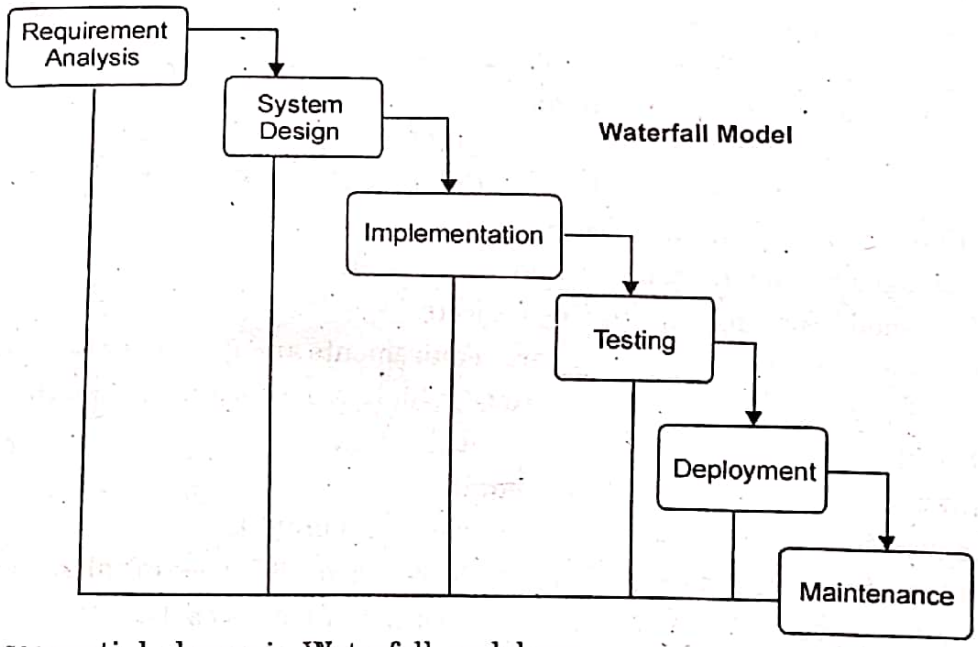
Q.2.(b) What is software life cycle? Discuss the generic waterfall model in detail. **(5)**

Ans. Software Development Life Cycle: Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

Generic waterfall model: 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.

The following illustration is a representation of the different phases of the Waterfall Model.



Waterfall Model

The sequential phases in Waterfall model are

- **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design:** The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation:** With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance:** There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Advantages

Some of the major advantages of the Waterfall Model are as follows–

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The major disadvantages of the Waterfall Model are as follows–

- 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 identifyin any technological or business bottleneck or challenges early.

Q.3. (a) What is Risk? What are the risk management activities? Is it possibl to prioritize the risk?

Ans. Risk: Risk is an expectation of loss, a potential problem that may or may n occur in the future. It is generally caused due to lack of information, control or time. possibility of suffering from loss in software development process is called a softwa risk. Identify the risk. Reduce the impact of risk.

Risk management activities: Risk is an inexorable and an unavoidable part o software development process, which constantly evolves throughout the course o project, affecting a project or software or both. Thus, it arise the necessity to deal a manage these risks in an efficient and effective manner.

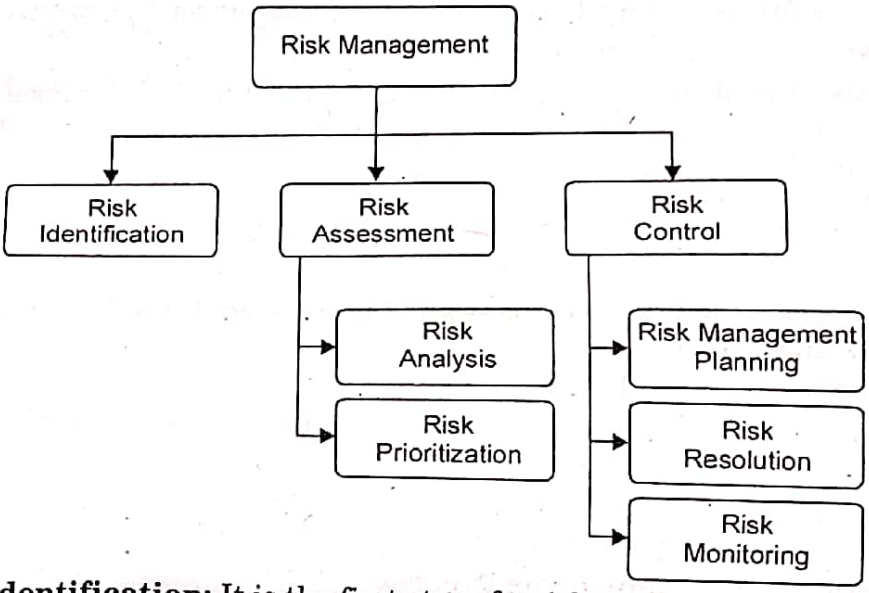
In the field of software engineering, risk management is a methodology or a mechanism, carried out throughout the development process to identify, manage and control risks evolved before and during the development process.

Basically, three types of activities are covered under the risk management process.

→ Risk Identification.

→ Risk Analysis.

→ Risk Control.



(a) Risk Identification: It is the first step of a risk management process, which involves the identification of potential risks that may affect a software product or a development project, and accordingly documenting them along with their characteristics.

It is a constant process, which is carried out throughout the development due to the fact that as the development process progresses, the more we get to know about the software product and based on it, we may able to explore and identify more unvisited or hidden risks.

Generally, this phase helps in identifying the two types of risks, product risk and project risk.

- **Product risk:** Risks pertaining to a software product or application, which may arise, due to its inefficiency, to function, desirably, to meet the expectation of the users.
- **Project risk:** These risks involve any sort of uncertain or unexpected event or action, which may likely to occur and degrade the progress of a project.

In this phase, usually client, stakeholders, business manager, project manager and test manager, collaborate and participates in brainstorming or small sessions, study and analyze the project documentation plan, etc., to make out the probable list of risks associated with the software development. Some commonly known techniques to identify risks may include risk templates, project retrospective, Failure Mode and Effect Analysis (FMEA), Failure Mode Effect and Criticality Analysis (FMECA), etc.

(b) Risk Assessment:

The next stage of a risk management process is risk analysis, which involves the assessment of the risks identified during the risk identification stage.

- This stage usually involves the analysis and prioritization of the risks, i.e. possible outcomes of each identified risk is being assessed based on which risks are categorized and accordingly, prioritized.

- Based on the degree of impact, possessed by each risk, they are being assigned severity levels, namely 'High', 'Medium' and 'low'. And based on their severity, they are prioritize i.e. High risks are considered as top priority whereas the low risk is regarded for the bottom most priority.

(c) Risk Control: During this stage, risks are managed, controlled and mitigated based on their priority so as to achieve the desired results. It is generally divided into three activities which may be seen below.

- **Risk Management Planning**: It involves a proper and effective plan to deal with the each identified risk.
- **Risk Resolution:** It refers to the execution of the plans, outlined during the risk management planning stage so as to either remove or fix identified risks.
- **Risk Monitoring:** It involves, regular monitoring and tracking, the development progress, in the direction, of resolving risk issues, which may include revaluation of the risks, their likelihood to occur, etc., and taking and implementing necessary & appropriate actions, wherever necessary.

Q.3. (b) Describe the various steps of requirements engineering. Is it essential to follow these steps? (5)

Ans. Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

(a) Feasibility study

- When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.
- Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.
- This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms o implementation, contribution of project to organization, cost constraints and as pe values and objectives of the organization. It explores technical aspects of the projec and product such as usability, maintainability, productivity and integration ability.
- The output of this phase should be a feasibility study report that should contai adequate comments and recommendations for management about whether or not th project should be undertaken.

(b) Requirement Gathering: If the feasibility report is positive towar undertaking the project, next phase starts with gathering requirements from the us Analysts and engineers communicate with the client and end-users to know their ide on what the software should provide and which features they want the software to inclu

(c) Software Requirement Specification: SRS is a document created by syst analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, exter interfaces, speed of operation, response time of system, portability of software acr various platforms, maintainability, speed of recovery after crashing, Security, Qual Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

(d) SRS should come up with following features:

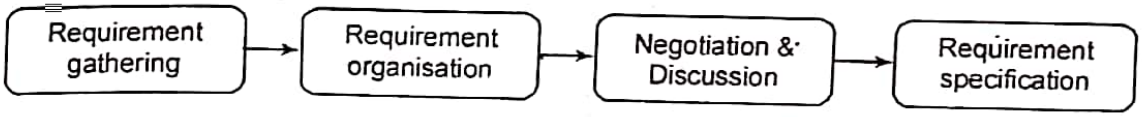
- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

- After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -
  - If they can be practically implemented
  - If they are valid and as per functionality and domain of software
  - If there are any ambiguities
  - If they are complete
  - If they can be demonstrated

Requirement Elicitation Process

- Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Q.4. (a) What are the size metrices? How function point metric is advantageous over LOC metric? Explain ? **(5)**

Ans. Size metrices: Measure can be defined as quantitative indication of amount, dimension, capacity, or size of product and process attributes. Metrics can be defined as quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project, and product.

Advantages of function point analysis:

Function Point Analysis provides the best objective method for sizing software projects, and for managing the size during development. Following are some of the many advantages that FPA offers.

(a) Helps Comparison: Since Function Points measures systems from a functional perspective they are independent of technology. Regardless of language, development method, or hardware/platform used, the number of FP for a system will remain constant. The only variable is the amount of effort needed to deliver a given set of FP ; therefore, Function Point Analysis can be used to determine whether a tool, an environment, a language is more productive compared with others within an organization or among organizations. This is a critical point and one of the greatest values of Function Point Analysis.

(b) Helps Monitor Scope Creep: Function Point Analysis can provide a mechanism to track and monitor scope creep. FP counts at the end of requirements, analysis, design, code, testing and deployment can be compared. The FP count at the end of requirements and/or designs can be compared to FP actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.

(c) Ease of Contract Negotiations: From a customer view point, Function Points can be used to help specify to a vendor, the key deliverables, to ensure appropriate levels of functionality will be delivered, and to develop objective measures of cost-effectiveness and quality. They are most effectively used with fixed price contracts as a means of specifying exactly what will be delivered. From a vendor perspective, successful management of fixed price contracts depends absolutely on accurate representations of effort. Estimation of this effort (across the entire life cycle) can occur only when a normalized metric such as the one provided by Function Points is applied.

Note: Here, my meaning of the word normalized metric is that Function Point truly accounts for the entire gamut of software development spreading across all the phases from Requirements through Testing. Whereas, LOC pertains to and is an outcome of only one of the phases.

(d) Handling Volatility: The advantage that Function Points bring to early estimation is the fact that they are derived directly from the requirements and hence show the current status of requirements completeness. As new features are added, the function point total will go up accordingly. If the organization decides to remove features or defer them to a subsequent release, the function point metric can also handle this situation very well, and reflect true state.

(e) Use of Historic Data: Once project size has been determined in Function Points, estimates for Duration, Effort, and other costs can be computed by using historic data. Since FP is independent of languages or tools, data from similar past projects can be used to produce consistent results,unlike Lines of Code data which is much tightly tied to languages requiring many other parameters to be taken into account.

(f) Availability of Empirical Formulae: Unlike lines of code, FP can be used more effectively to develop many predictive formulae such as defect potential, maintenance effort which can help pinpoint opportunities for improvement.

Q.4.(b) Discuss Cost estimation model(COCOMO) in detail. (6)

Ans. COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the **COCOMO** are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm's definition of organic, semidetached, and embedded systems:

1. Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.

3. Embedded – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes these Cost Drivers into account and **Detailed COCOMO** additionally accounts for the influence of individual project phases, i.e in

case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

Estimation of Effort: Calculations –

1. Basic Model: The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B
Organic	2.4	1.05
Semi Detached	3.0	1.12
Embedded	3.6	1.20

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

2. Intermediate Model: The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

3. Detailed Model: Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.