INTRODUCTION TO SOFTWARE ENGINEERING

Multiple Choice Type Questions

1. Structured systems development cycle approach includes which of the following characteristics? [WBUT 2013]
   a) dataflow b) top down
   c) data structure d) none of these
Answer: (c)

2. A prototype refers to [WBUT 2013]
   a) a working model of a proposed system b) the set of activities of a system
   c) both (a) and (b) d) none of these
Answer: (c)

3. The most important feature of spiral model is [WBUT 2013]
   a) requirement analysis b) quality management
   c) risk management d) configuration management
Answer: (c)

4. A COCOMO model is [WBUT 2013]
   a) common cost estimation model b) complete cost estimation model
   c) constructive cost estimation model d) none of these
Answer: (c)

5. MTTF is a measure of – [WBUT 2014]
   a) reliability b) maintainability c) cost of effort d) testability
Answer: (a)

6. Which is NOT a non-functional requirement? [WBUT 2014]
   a) efficiency b) reliability c) product features d) stability
Answer: (c)

7. Which model is generally used for developing GUI of a system? [WBUT 2014]
   a) spiral b) prototyping c) iterative waterfall d) evolutionary
Answer: (a)

8. Data hiding can be achieved by [WBUT 2014]
   a) Data Encapsulation b) Data Overloading
   c) Data Abstraction d) Polymorphism
Answer: (a)

9. According to COCOMO number of cost drivers is [WBUT 2015]
   a) 10 b) 15 c) 20 d) 14
Answer: (b)

10. Which form of software development model is most suited to a system where all the requirements are known at the start of a project and remain stable throughout the project? [WBUT 2015]
   a) Waterfall model b) Incremental model
   c) Evolution model d) Spiral model
Answer: (a)

11. DFD shows [WBUT 2017]
   a) the flow of data b) the processes
   c) the areas where they are stored d) all of these
Answer: (d)

12. Software maintenance includes [WBUT 2017]
   a) Setting preventing maintenance policy for servers
   b) Installation of software at site
   c) Designing of software for maintenance purposes
   d) Bug fixing
Answer: (d)

13. Prototype is a [WBUT 2017]
   a) Working model of existing system b) Mini model of existing system
   c) Mini model of processed system d) none of these
Answer: (a)

14. DFD balancing means [WBUT 2018]
   a) balancing of weight of processes
   b) must match the total number of bubbles
   c) must match the data flow at the next level of DFD
   d) None of the above
Answer: (c)

15. Software testing is the [WBUT 2018]
   a) process of demonstrating that errors are not present
   b) process of establishing confidence that a program does what it is supposed to do
   c) process of executing a program to show that it is working as per specifications
   d) process of executing a program with the intent of finding errors
Answer: (d)

Short Answer Type Questions

1. What are the drawbacks of classical waterfall model? How are they rectified in other variants of the classical waterfall model? **[WBUT 2013]**

OR,

What is SDLCM? What are the Disadvantages in Classical Waterfall Model? **[WBUT 2017]**

Answer:

1st Part:

A software development life cycle (SDLC) model is a conceptual framework describing all activities in a software development project from planning to maintenance. This process is associated with several models, each including a variety of tasks and activities.

2nd Part:

The classical model is the most widely used software development model. The name is suggested due to likeness of the model with the cascade of waterfalls. As each stage is signed off the next stage is proceeded with. End users are not involved in the development stage. It is the responsibility of the analysts and the programmers to understand the end-users requirements. This is a risky process with the waterfall model. The six different phases starting from the feasibility study are known as the development phases. At the end of the development the product will be delivered to the customer.

In the iterative model, revision of work on the earlier stages of the system can be made. It improves the performance of the system by gaining exposure at later stages of the system development. There can be various defects in the development process like; wrong understanding, incorrect design, quick and fix of bugs, hard coding etc. Once a defect is detected, the designer needs to move backward to detect the problem and redo the work in all the stages in sequence in the process.

2. What is feasibility study? Explain different types of feasibility study? **[WBUT 2013]**

Answer:

1st Part;

A feasibility study looks at the viability of an idea with an emphasis on identifying potential problems and attempts to answer one main question: Will the idea work and should we proceed with it?

Before we begin writing our business plan we need to identify how, where, and to whom we intend to sell a service or product. We also need to assess our competition and figure out how much money you need to start our business and keep it running until it is established.

Feasibility studies address things like where and how the business will operate. They provide in-depth details about the business to determine if and how it can succeed, and serve as a valuable tool for developing a winning business plan.

The information we gather and present in our feasibility study will help us:
List in detail all the things we need to make the business work;
Identify logistical and other business-related problems and solutions;
Develop marketing strategies to convince a bank or investor that our business is worth considering as an investment; and
Serve as a solid foundation for developing our business plan.

2nd Part:

Technical Feasibility

- Study of available technology to implement the system and the suggested alternatives.

Economic Feasibility

- It is the financial study related to the technical feasibility. The cost-effectiveness of the system – do the benefits outweigh the costs? Calculate the ROI (Return on investment) on present value method and also on discounted money value method to arrive at the money value.

Social/Operational Feasibility

- Legal Feasibility – Study of legal requirements related to technical feasibility. If there is any conflict about the system and legal procedures e.g. Data disclosure and privacy Act.
- Operational Feasibility – It is also known as the study of the social feasibility. It studies the impacts on the social factors on implementation of the system and how the work practices and procedures will be able to support the new system.
- Schedule feasibility – It is the time study about the proposed system. How long it will take to develop and monitoring of the proposed time frame.

3. Consider the size of an organic type s/w product that has been estimated as 32,000 lines of source code. The average salary of s/w developers is 15,000 p.m. Determine the effort required to develop the s/w, the nominal development time and the cost to develop the s/w, the nominal development time and the cost to develop the product. **[WBUT 2013]**

Answer:

Effort= $2.4(KLOC)^{1.05}$ PM $= 2.4 \times (32)^{1.05}$ PM
$= 2.4 \times 38.05462 = 91.331$ PM

Tdev = $2.5(Effort)^{0.38}$ months $= 2.5 \times (91.331)^{0.38}$
$= 2.5 \times 5.559 = 13.898$ months

Cost required to develop the product $= 13.898 \times 15{,}000$ /- = Rs. 2,08,481/-

4. a) What is phase containment of errors? **[WBUT 2013]**
b) Describe structured analysis and structured design.

Answer:

a) Detection and correction the error within the segment or phase where it actually lies. E.g. the design error should be spotted and rectified within the design phase rather than fixing it in the coding phase. This is the objective behind development of the iterative waterfall model and which saves time and money during the system development. A periodic review is necessary for detecting phase containment of errors.

b) Structured Analysis and Design Technique (SADT) is a diagrammatic notation designed specifically to help people describe and understand systems. It offers building blocks to represent entities and activities, and a variety of arrows to relate boxes. These boxes and arrows have an associated informal semantics. SADT can be used as a functional analysis tool of a given process, using successive levels of details. The SADT method not only allows one to define user needs for IT developments, which is often used in the industrial Information Systems, but also to explain and present an activity's manufacturing processes and procedures.

The SADT supplies a specific functional view of any enterprise by describing the functions and their relationships in a company. These functions fulfill the objectives of a company, such as sales, order planning, product design, part manufacturing, and human resource management. The SADT can depict simple functional relationships here and can reflect data and control flow relationships between different functions.

5. What are Halstead's metrics? [WBUT 2013]

Answer:

Halstead's theory of software science is one of the best known and most thoroughly studied composite measures of (software) complexity. Software science proposed the first analytical laws for computer software.

Software science assigns quantitative laws to the development of computer software, using a set of primitive measures that may be derived after code is generated or estimated once design is complete these follow.

n 1- The number of distinct operators that appear in a program.

n2- The number of distinct operands that appear in a program.

N1 - The total number of operator occurrences.

N 2- The total number of operand occurrences.

Halstead uses these primitive measures to develop expressions for the overall program length, potential minimum volume for an algorithm, the actual volume (number of bits required to specify a program), the program level (a measure of software complexity), the language level (a constant for a given language) and other features such as development effort, development time and even the projected number of faults in the software.

Halstead shows that length N can be estimated

$N = n1 \log_2 n1 + n2 \log_2 n2$ ... (1)

and program volume may be defined

$V = N \log_2 (n1 + n2)$ ... (2)

V will vary with programming language and represents the volume of information (in bits) required to specify a program.

Theoretically, a minimum volume must exist for a particular algorithm. Haistead defines a volume ratio L as the ratio of volume G the most compact form of a program to the volume of the actual program. In actuality, L must always be less than 1. In terms of primitive measures, the volume ratio may be expressed as

$L = 2/n1 \times n1/N2$

Halstead's work is amenable to experiment verification and large body of research has been conducted to investigate software science.

6. a) What are 'baselines' with respect to software configuration management?
b) What is the necessity of software configuration management in developing a software? [WBUT 2014]

Answer:

a) Base line is a one or more software configuration items that have been formally reviewed and agreed upon and serve as a basis for further development. Whereas configuration is a collection of all the elements of a baseline and a description of how they fit together. The basic purpose of Baseline provides,

1) Measurable progress points within the SDLC
2) A basis for change control in subsequent project phases
3) A constant reference for future work
4) Intermediate and final points for assessing the health for purpose of project work products.

b) According to Barry Boehm "Producing software from a specification is like walking on water - it's easier if it's frozen.

Baseline is a reference point in the SDLC marked by the completion and formal approval of a set of predefined work products. The goal of a baseline is to decrease a project's weakness towards wild changes by fixing and formally change controlling various key deliverables (configuration items) at critical points in the SDLC. Baselines are also used to identify the aggregate of software and hardware components that make up a specific release of a system.

7. A project was estimated to be 200 KLOC. Calculate the effort development time, average staff size and productivity level for
i) Organic
ii) Semi-detached modes. [WBUT 2014]

Answer:

i) $\text{Effort} = 2.4(\text{KLOC})^{1.05} = 2.4(200)^{1.05} = 625.5942$

$\text{Tdev} = 2.5(\text{Effort})^{0.38} = 2.5(625.5942)^{0.38} = 28.88$ months

Average Staff Size = Effort/Time Schedule= 21.66

Productivity level = [KLOC/EFFORT] DSI/MM=200/625.5942

= 0.3197 DSI/MM

Effort = $3.0(KLOC)^{1.12} = 3.0(200)^{1.12}$ =1133.1172
Tdev = $2.5(Effort)^{0.35} = 2.5(1133.1172)^{0.35}$=29.30 months
Average Staff Size = Effort/TimeSchedule= 38.67
Productivity level = [KLOC/EFFORT] DSI/MM=200/1133.1172
= 0.1765 DSI/MM

8. The size of an organic type software product has been estimated to be 1,00,000 lines of source code. The average salary of software developers is Rs. 10,000/- per month. Determine the effort required to develop software product, the nominal development time and the cost to develop the product. [WBUT 2015]

Answer:

Using the cost drivers for organic model of the Cocomo
Effort (E) = $2.4(100)^{1.05} = 2.4 \times (125.8925)$ PM = 302.1420 PM
Tdev (D) = $2.5(302.1420)^{0.38} = 2.5(8.7595)$M = 21.8927
= 22 months (Approx.)
Cost of developer is= 22x (10000) =Rs. 2,20,000/-

9. a) Explain the phases of Spiral Model with advantages and disadvantages. [WBUT 2015]

OR,

Explain Spiral Model for software development with a diagram. [WBUT 2017]
b) Explain the advantages and disadvantages of prototype model. [WBUT 2015]

Answer:

a) There are several advantages and disadvantages of spiral model, which should be considered before finalizing on spiral model implementation. The overview of steps in developing a spiral life cycle model can be given as below.

- ***Step 1:*** The requisites of the new system are described in depth, by consulting all the users of the existing model and an introductory system design is prepared for new model or system.
- ***Step 2:*** First archetype is built up with features close to the final system, followed by creating second type.
- ***Step 3:*** Creating second prototype involves evaluating the performance of the first and describing the requisites of the second prototype, followed by building and testing the second architecture.
- ***Step 4:*** The discrepancies in the estimated running cost are evaluated and the efficiency of the new prototype is tested to find out if the new model meets the expectations of the customer.
- ***Step 5:*** The steps in creating new prototype are repeated till the new prototype fulfills all the demands or requisites desired by the customer.
- ***Step 6:*** Maintenance of the new model is done to avoid break down, till it is assured that the new system is working smoothly.

Just like any other system or model, a client should evaluate spiral model advantages and disadvantages. Let's go through these quickly.

Spiral Model Advantages

Repeated or continuous development helps in risk management. The developers or programmers describe the characteristics with high priority first and then develop a prototype based on these. This prototype is tested and desired changes are made in the new system. This continual and steady approach minimizes the risks or failure associated with the change in the system.

Adaptability in the design of spiral model in software engineering accommodates any number of changes, that may happen, during any phase of the project.
Since the prototype building is done in small fragments or bits, cost estimation becomes easy and the customer can gain control on administration of the new system.

As the model continues towards final phase, the customer's expertise on new system grows, enabling smooth development of the product meeting client's needs.

Spiral Model Disadvantages

The following can be summarized as the disadvantages of the spiral model.
Spiral models work best for large projects only, where the costs involved are much higher and system pre requisites involves higher level of complexity.
Spiral model needs extensive skill in evaluating uncertainties or risks associated with the project and their abatement.

Spiral models work on a protocol, which needs to be followed strictly for its smooth operation. Sometimes it becomes difficult to follow this protocol.
Evaluating the risks involved in the project can shoot up the cost and it may be higher than the cost for building the system.
There is a requirement for further explanation of the steps involved in the project such as breakthrough, blueprint, checkpoints and standard procedure.

Spiral model serves as the best option for businesses with volatile business goals but where there is a need for a prototype to handle the complexities in the business procedures. This was about spiral model advantages and disadvantages and spiral model development steps. I hope this article has helped you with spiral model pros and cons.

b) Advantages of Prototype model:

The system prototypes have the following benefits:

1. Misunderstandings are detected at early stages
2. User will notice incomplete or inconsistent requirements.
3. Toy model can be built quickly to demonstrate systems
4. Fits top-down building and testing strategies
5. It can be used for training before the system is finished

6. The prototype may be used as a basis for writing the specification for a production-quality system.

The benefits of using prototyping in the software process are:

1. Improved system usability;
2. Improved design quality
3. Improved maintainability
4. Reduced development effort

Disadvantages of Prototype model:

Some of the disadvantages of the prototyping model are:

1. User may get bored and frustrated due to repetition of the process
2. Sometimes system becomes quite unreliable, if not throwaway type.
3. The size of the prototype is limited
4. A layered style of waterfall model is viewed as weaknesses in one direction only
5. It is expensive and repetition of work is involved.

10. Discuss the characteristics of a good SRS document. **[WBUT 2016]**

OR,

Describe the various steps of requirements engineering. Is it essential to follow these steps? **[WBUT 2016]**

Answer:

A good SRS should contain:

i) Interfaces
ii) Functional Capabilities
iii) Performance Levels
iv) Data Structures/Elements
v) Safety
vi) Reliability
vii) Security/Privacy
viii) Quality
ix) Constraints and Limitations

11. List three common types of risks that a typical software project might suffer from. **[WBUT 2016]**

Answer:

Refer to Question No. 4(b) of Long Answer Type Questions.

12. Draw the use case diagram of a library management system. **[WBUT 2016]**

Answer:

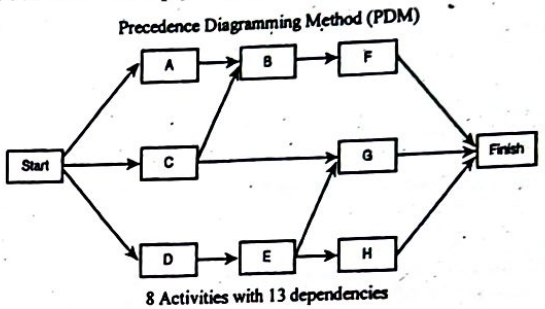
Refer to Question No. 3(a) (2nd Part) of Long Answer Type Questions.

13. Why Project Planning is needed? Draw the diagram for precedence ordering among planning activities. **[WBUT 2017]**

Answer:

Project planning is part of project management, which relates to the use of schedules such as Gantt charts to plan and subsequently report progress within the project environment.

Project planning is often used to organize different areas of a project, including project plans, workloads and the management of teams and individuals. Project planning is inherently uncertain as it must be done before the project is actually started. Therefore the duration of the tasks is often estimated through a weighted average of optimistic, normal, and pessimistic cases. Float or slack time in the schedule can be calculated using project management software. Then the necessary resources can be estimated and costs for each activity can be allocated to each resource, giving the total project cost. At this stage, the project schedule may be optimized to achieve the appropriate balance between resource usage and project duration to comply with the project objectives.



8 Activities with 13 dependencies

The PDM diagram in the figure above shows eight planning activities, labeled A–H. The arrows show how some activities are dependent on other activities. E.g. activity B cannot start until activities A and C are complete. To show this dual dependency, an arrow can be drawn from A to B and another arrow from C to B.

There are four types of dependencies with a PDM diagram:

- **Finish-to-start (the most common dependency type)**—The successor activity's start depends on the completions of the successor activity.
- **Finish-to-finish**—The completion of the successor activity depends on the completion of the predecessor activity.
- **Start-to-start**—The start of the successor activity depends on the start of the predecessor activity.
- **Start-to-finish**—The completion of the successor activity depends on the start of the predecessor activity.

14. Explain Empirical Cost Estimation Techniques. **[WBUT 2017]**

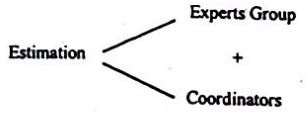
Answer:
Empirical estimation technique is based on the data taken from the previous project and some based on guesses and assumptions. There are many empirical estimation techniques but most popular are

a) Expert Judgement Technique
b) Delphi Cost Technique

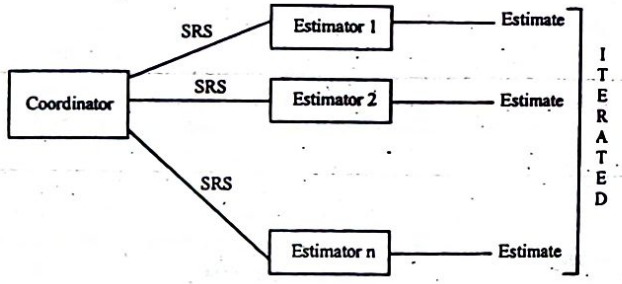
Expert judgement technique:
An expert makes an educated guess of the problem size after analyzing the problem thoroughly. Expert estimate the cost of different components that is modules and sub modules of the system.

Disadvantages:
Human error, considering not all factors and aspects of the project, individual bias, more chances of failure.
Estimation by group of experts minimizes 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 completes 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

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. After the completion of several iterations of estimation, the coordinator takes the responsibility of compiling the result and preparing the final estimates.

15. What are the different of information elicitation? **[WBUT 2018]**
Answer:
Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need. There are a number of requirements elicitation methods. A few of them are listed below –

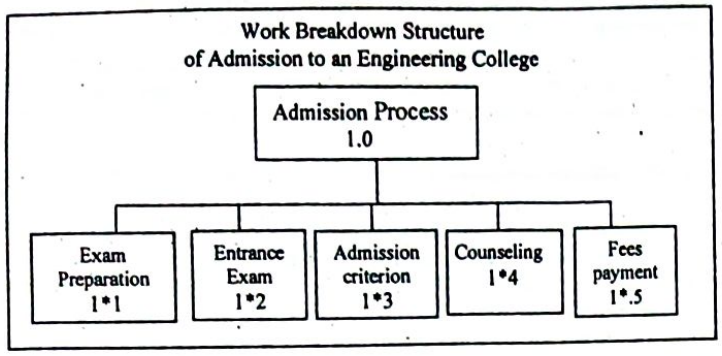
1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users and the customer involved.

16. Develop a work breakdown structure specification for showing the process of admission to an engineering college. Assume major phase as exam preparation, entrance exam, admission criterion and counseling and fees payment. Also write the output of each major task performed. **[WBUT 2018]**
Answer:
The efficiency of a work breakdown structure can determine the success of a project. The WBS provides the foundation for all project management work, including, planning, cost and effort estimation, resource allocation, and scheduling. Therefore, one should take creating WBS as a critical step in the process of project management.

Outputs

Level	WBS	Task Description
1	1.0	Admission Process
1	1.1	Exam Preparation
1	1.2	Entrance Exam
1	1.3	Admission criterion
1	1.4	Counseling
1	1.5	Fees payment



17. 'A good software should have high cohesion but low coupling' – Explain. **[WBUT 2018]**

Answer:

One of the most important goals of object oriented design is to have high cohesion classes and loose coupling between these classes.

Coupling refers to links between separate units of a program. In object oriented programming, if two classes depend closely on many details of each other, we say they are tightly coupled.

Coupling is a measure of the interdependence between classes. If every object has a reference to every other object, then there is tight coupling, and this is undesirable. Because there's potentially too much information flow between objects. Loose coupling is desirable. It means that objects work more independently of each other. Loose coupling minimizes the "ripple effect" where changes in one class cause necessity for changes in other classes.

Cohesion, on the other hand, refers to the number and diversity of tasks that a class is designed for. If a class is responsible for a few related logical tasks, we say it has high cohesion.

Cohesion is a measure of strength of the association of variables and methods within a class. High cohesion is desirable because it means the class does one job well. Low cohesion is bad because it indicates that there are elements in the class which have little to do with each other. Modules whose elements are strongly and genuinely related to each other are desired. Each method should also be highly cohesive. Most methods have only one function to perform. Don't add extra instructions into methods that cause it to perform more than one function.

Low coupling makes it possible to:

- Understand one class without reading others
- Change one class without affecting others
- Thus: improves maintainability

High cohesion makes it easier to:

- Understand what a class or method does
- Use descriptive names
- Reuse classes or methods

[WBUT 2018]

18. What are the major components of SRS?

Answer:

The basic issues an SRS must address are:

- Functionality.
- Performance.
- Design constraints imposed on an implementation.
- External interfaces.

Design Control Process

1. Which outputs should be produced from the given inputs?
2. Relationship between the input and output.
3. A detailed description of all the data inputs and their source, the units of measure.
4. The range of valid inputs.

Design Constraints

1. Standards that must be followed.
2. Resource limits & operating environment.
3. Reliability
4. Security requirement
5. Policies that may have an impact on the design of the system.

Standards Compliance:

This specifies the requirements for the standards that the system must follow.

Hardware Limitations:

The software may have to operate on some existing or predetermined hardware thus imposing restrictions on the design.

Reliability and 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.

Security:

Security requirements are particularly significant in defense systems and many database systems. Security requirements place restrictions on the use of certain commands, control access to data, 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.

External Interface Requirements:

- All the possible interactions of the software with people, hardware and other software should be clearly specified. For the user interface, the characteristics of each user interface of the software product should be specified. User interface is becoming increasingly important and must be given proper attention. A preliminary user manual should be created with all use commands, screen formats and explanation of how the system will appear to the user, and feedback and error message.

Like other specifications these requirements should be precise and verifiable. So a statement likes "the system should be no longer than six characters" or command names should reflect the function they perform used. If the software is to execute on existing hardware or on predetermined hardware, all the characteristics of the hardware, including memory restrictions, should be specified. In addition, the current use and load characteristics of the hardware should be given.

19. What do you mean by functional and non-functional requirements?
[MODEL QUESTION]

Answer:

Functional Requirements define *what* the system needs to do, Functional requirements are what the business users expect the software to 'do'; what tasks they wish it to perform. (e.g. write paychecks, calculate launch date for lunar orbit, do hours to gross calculation for movie studios, etc.).

Whereas non Functional Requirements describe *how* the system will do it. The functional requirement is to produce a pay slip for an employee, the non-functional requirement is to print a pay slip per employee (up to, say, 10,000) in, say, 48 hours elapsed. The non functional requirement may be one of the Critical Success Factors, it may even be the prime driver for the development, but it is still non-functional. Non functional requirements have also been called the 'ilities' because they are most simply expressed like this:

1. usability
2. reliability
3. interoperability
4. scalability
5. security

Functional Requirements are either met or not met. Non-functional requirements tend to be things that you can measure.

20. Explain the phases of Rapid Prototype Life Cycle Model with advantages and disadvantages.
[MODEL QUESTION]

Answer:

Rapid Prototyping Model

- Emphasis is on creating a prototype that looks and acts like the desired product in order to test its usefulness

- Develop a system with reduced capability
- Present to client for approval
- Once the prototype is approved, it is discarded and the "real" software is written.
- Develops specification with better understanding
- Exactly like the Spiral Model, where a prototype or only "shadow" of the real software is made, where the system (during implementation stage) is not that graphically perfect but features are functioning well for testing purposes
- Only difference is, in Rapid Prototyping, it requires client approval prior to the building of the "real" software.

Advantages of Prototyping

Users are actively involved in the development

It provides a better system to users, as users have natural tendency to change their mind in specifying requirements and this method of developing systems supports this user tendency.

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 as the system is mode side by side.

Quicker user feedback is available leading to better solutions.

Disadvantages

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.

Long Answer Type Questions

1. a) Give the guideline formulated in IEEE 830 for writing SRS document.
b) Assuming standard activities in a Library, write SRS according to IEEE 830.
[WBUT 2013]

Answer:

a) *How to write the SRS documentation, following IEEE Std. 830*

1. General

1.1 All documents should have a title page (to include information such as: title of the project, course name and number, team members, place, date, and other relevant information).

1.2 Table of Contents normally makes a lot of sense, so should be included as well

1.3 Number all sections in the document.

1.4 Any reference correctly included in Section 1.4 should be written as follows:

For books, reports, theses, standards, manuals, etc.:

[number] NameOfAuthor(s), TtitleOfWork, Publisher, Place, Date

(if authors' or editors' names are not available, it can start with a title of the

name of the Publisher)
For papers/articles:
[number] NameOfAuthor(s), TtitleOfWork, JournalName, VolumeNumber, IssueNumber, PageNumbers (pp. first-last), Year (or Month and Year)
For papers in Conference Proceedings:
[number] NameOfAuthor(s), TtitleOfWork, Phrase "Proceedings of the" Conference Name, Place, Date[, Publisher, Place Date]
For websites:
[number] AuthorNames(s), TitleOfWork, Company's Name, Place, Date, URL
Note. For names of authors never use full first names, only initials!

1.5 All references from Section 1.4 have to be referred to in the text (using [number] notation)
1.6 Do not end section titles with colons.
1.7 Every figure/diagram should have a caption (number and titile). Place it underneath the figure/diagram.
1.8 Every table should have a number and title, placed above the table.
1.9 "Shall/Must" phraseology should not be used in unless it is requirement. This means that normally it is not used in sections 1 or 2.

2. Writing "Introduction"

- In Section 1.1 "Purpose", describe the purpose of this document, not the purpose of the software being developed.
- In Section 1.2 "Scope", describe the scope of this document, not the scope of the software being developed.
- In Section 1.5 "Overview", provide an overview of this document, not the overview of the software being developed.
- Definitions, in Section 1.4, should be write using the following template:
  word_defined. <in lower case, ended with a dot "."> Followed by a definition.
  For example:
  user. The person, or persons, who operate or interact directly with the product.

3. Writing "Overall Description"

- a Context Diagram is mandatory.
- other important characteristics are: Product Perspective, product Functions, User Characteristics, Constraints, Assumptions and Dependencies.

4. Writing "Specific Requirements"

- Never specify the Operating System or Language in the SRS, unless the customer demands doing so. These are strictly implementation issues, and well designed software can be implemented in any specific programming language to run under any specific operating system on any specific hardware platform.
- Specific Requirements Section should be split into:
  * "External Interfaces" derived from the Context Diagram
  * "Functional Requirements" that should be further split into "Input Requirements"

(related to user inputs, commands, etc.), "Output Requirements" (mostly related to the GUI), "Input/Output Requirements" (if they cannot be separated), and "Processing
Requirements"
* "Non-Functional Requirements", such as performance, reliability, safety, security, etc.
* "Design Constraints", normally related to software and hardware limitations (OS, platform, stand-alone or networked, network protocols, standards, etc.);
* "Database Requirements" – can be combined with "Design Constraints".

- Use Case Diagrams have to be included in most sections, specifically in the "Functional Requirements" section. Several Use Case Diagrams have to be presented, including specific scenarios, how the system will respond to certain user/operator requests or commands, or network behavior.

5. Other

- End your SRS document by the following line (centered across the page):

b) SRS of LIBRARY MANAGEMENT SYSTEM

TABLE OF CONTENTS

1. Introduction

1.1 Purpose: The purpose of this document is to describe the LMS System. It consists of the functional, behavioral and non-functional requirements of the project with system engineers and designers guidelines.

1.2 Scope

LMS is principally transforming the manual system into a web enabled application so that the users know the details about availability books, time for borrowing, and account details. It will work as a complete user interface for library management process and library usage

1.3 Projected Audience, acronyms and abbreviations:

The intended users of this document are the developers, testers, library owners, managers, and coordinators.

1.3.1 Acronyms and Abbreviations Acronym	Meaning
MS SQL	Microsoft Structured Query Language
ASP	Active Server Pages
ISBN	International Standard Book Number
IEEE	Institute of Electrical and Electronics Engineers

Overall Description

Product Perspective: LMS is a replacement for the ordinary LMS based on paper work for recording books.

Product Functions

- Admin should be able to insert, modify and delete books.
- Can accept or reject a new user according to the library policy or payment methods.
- Increase the period for borrowing a book for specific type or group of users.
- Can get the information (status report) of any member who has borrowed a book.
- Add and edit book categories and arrange books by categories.
- Add and edit authors and publishers information.
- Can send lateness warnings to people who have exceeded deadline date.
- Can record books returned by users.

2.2.1 Normal Users (Library Members)

- The member should be provided with the updated information about the books catalog.
- Members are given an access to check their account's information and change it.
- Members have the ability to search through books by subject, title, authors or any information related to the book.
- Can extend the period of borrowing books according to the library policy.
- The customer may suggest a book to be brought to the library book collection.

2.3 Operating Environment

The LMS will operate on web site over browsers like Microsoft Internet Explorer, Google Chrome, Mozilla Firefox with Flash Player and JavaScript.

2.4 User Characteristics

Users of this LMS are members, librarians and the administrators who maintain the website. Members and librarians are assumed to have basic knowledge of computers and Internet browsing. Administrators of the system should have more knowledge of internal modules of the system and are able to rectify small problems that may arise due to disk crashes, power failures and other catastrophes.

2.5 Design and Implementation Constraints

- The information of all users, books and libraries must be stored in a database that is accessible by the website.
- MS SQL Server will be used as SQL engine and database.
- The Online Library System is running 24 hours a day.
- Users may access from any computer that has Internet browsing capabilities and an Internet connection.
- Users must have their correct usernames and passwords to enter into their online accounts and do actions.

2.6 Assumptions and Dependencies

- The product needs the following third party products.
- Microsoft SQL server to store the database.
- ASP.net develops the Product.

3. External Interfaces Requirements

3.1 User Interfaces and Login Interface

In case the user is not registered yet, he can enter the details and register. Which asks the user to type his username and password .If the user entered either his username or password incorrectly then an error message occurs.

Search:

The member or librarian can enter the type of book he is looking for and the title he is interested in them, and then can be searched for the required book by entering the book name.

Categories view: Categories view shows the books categories view with ability to Liberian to add/edit or delete category from the list.

Librarian's Control Panel

This control panel will allow librarians to add, confirm, or remove users; add, edit, or remove a medium. And manage lending options.

3.2 Hardware Interfaces

Only the recommended configuration (basic requirements of a computer system) no other specific hardware is required to run the software.

3.3 Software Interfaces

- ✓ Browser to load and view the web pages
- ✓ Operating System

4. Functional Requirements

4.1.1 Librarian

Insert book:
This action is done to add new book to library book collection
Delete / modify book: This event is to delete an existing book or modify its information.
Delete member: Admin can delete a member due to some specific rules.
Return book: Admin should confirm the return of books borrowed by users

4.1.2 Normal User

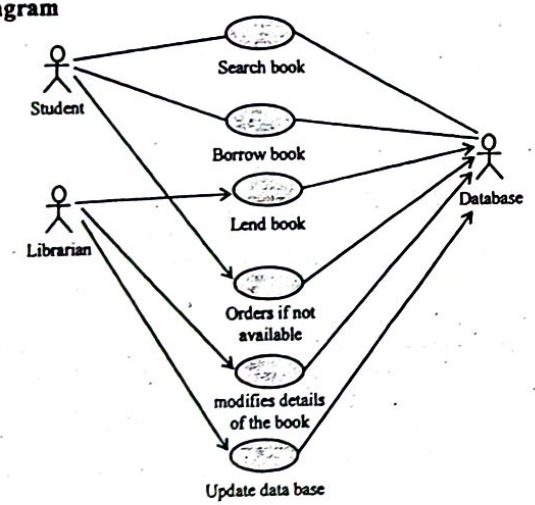
Register:
When new user enters for the first time then he has to register
Extending borrowing deadline:
Member can extend the borrowing time to some limit decided by Admin

4.1.3 Common Functions

Login:
Both Admin and members must be logged in before they modify any information
Search for book:
When user or admin wants to search on some book by name, author or subject etc.

5. Behavioral Requirements

USE CASE Diagram



6. Non-functional Requirements

Error handling

- Library Management System shall handle expected and non-expected errors in ways that prevent loss in information and long downtime period.

Performance Requirements

- The system shall accommodate high number of books and users without any fault.

Safety Requirements

- System use shall not cause any harm to human users

Security Requirements

- System will use secured database
- Normal users can just read information but they cannot edit or modify anything except their personal and some other information.
- System will have different types of users and every user has access constraints

2. Do a comparative study among the following CoCoMo models: [WBUT 2013]
a) Basic
b) Intermediate
c) Complete
d) COCOMO2.

Answer:

a) Constructive Cost Model (**COCOMO**) use cost drivers for estimation proposed by Barry Bohem, which is categorized as algorithmic methods. There are three forms of the constructive cost model:

1. Basic COCOMO which gives an initial rough estimate of man months and development time.
2. Intermediate COCOMO which gives a more detailed estimate for small to medium size projects.
3. Complete COCOMO which gives a more detailed estimate for large projects.

- **BASIC COCOMO**
  - ✓ It is used for relatively small project.
  - ✓ Only a few cost drivers are associated.
  - ✓ Cost drivers depend upon project size mainly.
  - ✓ Useful when the team size is small, i.e. small staff.

The effort (E) and schedule (S) of the project are calculated as follows

- ➢ Effort E = a * (KDSI) b * EAF Where KDSI is number of thousands of delivered source instructions a and b are constants, may vary depending on size of the project .
- ➢ Schedule S= c * (E) d where E is the Effort and c, d are constants.
- ➢ EAF is called Effort Adjustment Factor which is 1 for basic cocomo , this value may vary from 1 to 15.

The basic cocomo gives the magnitude of cost of the project. It varies depending upon size of the project. The various classes of software projects are

- Organic mode projects :
  - ✓ Used for relatively smaller teams.
  - ✓ Project is developed in familiar environment.
  - ✓ There is a proper interaction among the team members and they coordinate their work.
  - ✓ Bohem observed E=2.4(KDSI)1.05 E in person-months.
  - ✓ And S=2.5(E)0.38.
- Semi-detached mode projects :
  - ✓ It lies between organic mode and embedded mode in terms of team size.
  - ✓ It consists of experienced and inexperienced staff.
  - ✓ Team members are unfamiliar with the system under development.
  - ✓ Bohem observed E=3(KDSI)1.12 E in person-months.
  - ✓ And S=2.5(E)0.35.
- Embedded mode projects :
  - ✓ The project environment is complex.
  - ✓ Team members are highly skilled.
  - ✓ Team members are familiar with the system under development.
  - ✓ Bohem observed E=3.6(KDSI)1.20 E in person-months.
  - ✓ And S=2.5(E)0.32.

b) INTERMEDIATE COCOMO
- ✓ It is used for medium sized projects.
- ✓ The cost drivers are intermediate to basic and advanced cocomo.
- ✓ Cost drivers depend upon product reliability, database size, execution and storage.
- ✓ Team size is medium.

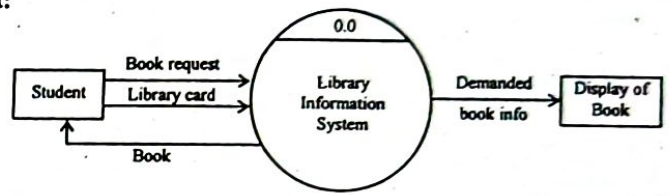
COMPLETE COCOMO
- ✓ It is used for large sized projects.
- ✓ The cost drivers depend upon requirements, analysis, design, testing and maintenance.
- ✓ Team size is large.

) COCOMO II
- COCOMOII was developed in 1995
- It could overcome the limitations of calculating the costs for non-sequential, rapid development, reengineering and reuse models of software.
- It has 3 modules
  - ✓ Application composition: good for projects with GUI interface for rapid development of project.
  - ✓ Early design: Prepare a rough picture of what is to be designed. Done before the architecture is designed.
  - ✓ Post architecture: Prepared after the architecture has been designed.

COCOMO II calculation
- ✓ In COCOMO II the constant value b is replaced by 5 scale factors.
- ✓ Effort (E) is calculated as follows:
  $E = a * (KDSI)^{sf} * \pi (EM)$
- ✓ Where a is constant, sf is scaling factor, EM is Effort Multiplier (7 for Early design, 17 for Post architecture).

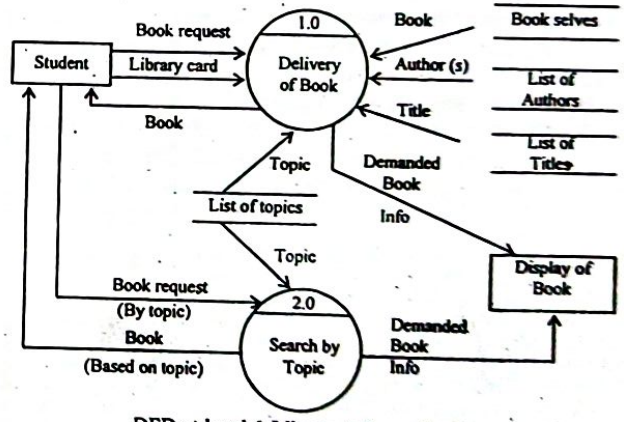
3. a) Draw the context diagram and Level-1 DFD for Library management system. Draw also USE-CASE diagram for this system.
b) What do you mean by balancing of DFD? Explain with a suitable example.
[WBUT 2014]

Answer:
a) 1st Part:



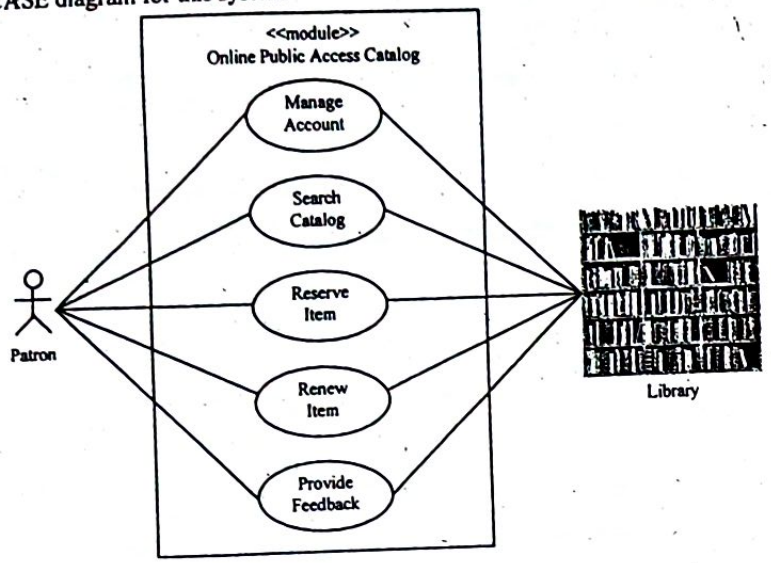
DFD at level-0 Library Information System

- Here inside 0 level data flow diagram, there will be one process called library management system
- There are many entities that directly interact with this system
- For example Student can issue a book a book, pays fine, get library card, issue book as well as request or return a book
- Demanded book will be displayed inside this system.



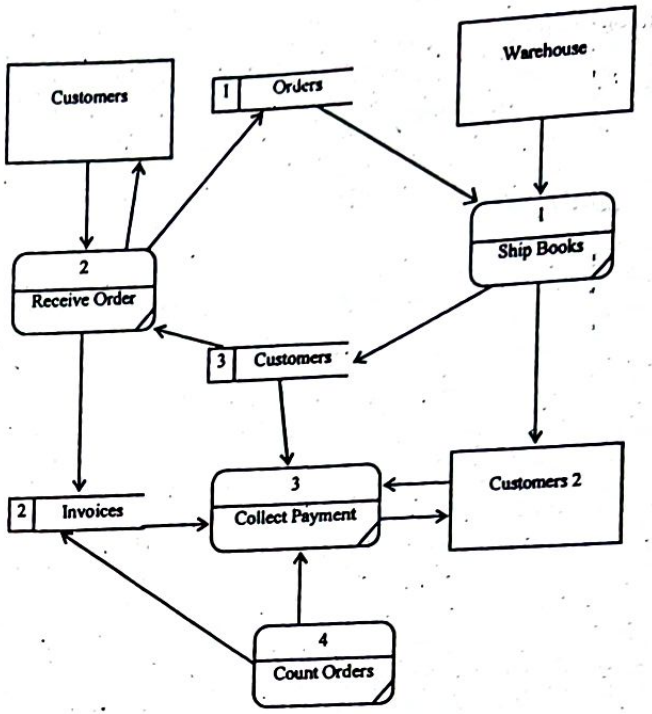
DFD at level-1 Library Information System

2nd Part:
USE-CASE diagram for this system



b) When decomposing a DFD, you must conserve inputs to and outputs from a process at the next level of decomposition.
The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing a DFD.

Process decomposition organize overall DFD in a series of levels so that each level provides successively more detail about a portion of the level above it. The goal of the balancing feature is to check the system internal consistency. When a process is decomposed, it helps to initialize the sub diagram, the objects from the upper-level to link to the sub-process. The following example shows a top level DFD, which is going to decompose the Collect Payment process:



4. a) What is SRS? Write the features of SRS. **[WBUT 2014]**

OR,

What do you mean by the term 'requirement'? Explain the process of determining the requirements for a software based system. **[WBUT 2016]**

b) What is Risk? Why Risk Analysis is done? **[WBUT 2014]**

Answer:

a) Software Requirement Specification (SRS) document usually contains a software vendor's understanding of a customer's software requirements. This document ensures that the software vendor and the customer are in agreement as to the features required in the software system being built. SRS is created after the initial requirement elicitation phase in which Software vendor interacts with the customer to understand the software needs. Usually SRS documentation is prepared by a business analyst who has some technical background.

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.

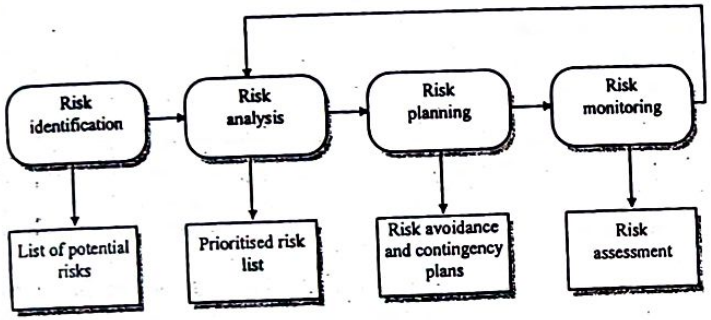
Characteristics of an SRS

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

b) High degree of precession and cost is involved in a software project so as the risk. Thus risk dictates assessment and analysis. Two activities are involved in the risk analysis.

1. Uncertainty
2. Loss

However, small and medium size software projects are being managed informally and on an Ad hoc basis. The time spent in identifying, analyzing, and managing risk pays itself back in many ways. Diagrammatically the risk management process may be presented as follows:



Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.

- A risk is a probability that some adverse circumstance will occur;
- Project risks affect schedule or resources;
- Product risks affect the quality or performance of the software being developed;
- Business risks affect the organisation developing or procuring the software.

The risk management process includes:

- Risk identification
- Identify project, product and business risks: Risk analysis
- Assess the likelihood and consequences of these risks : Risk planning
- Draw up plans to avoid or minimise the effects of the risk: Risk monitoring
- Monitor the risks throughout the project.

5. a) What is software engineering?
b) Discuss the software engineering process.
c) Explain waterfall model in detail with the help of diagram. State its advantage and also its limitations. **[WBUT 2016]**

Answer:

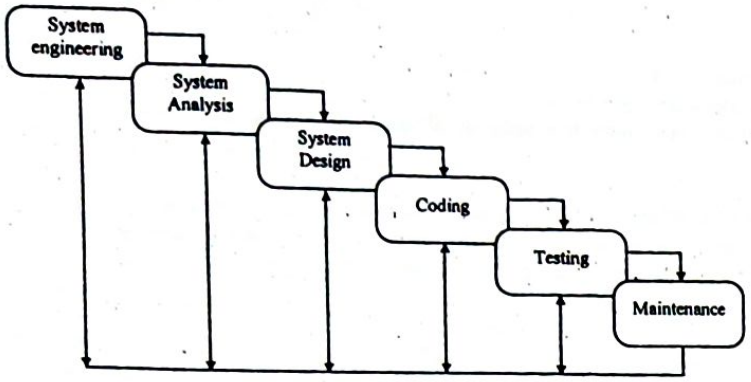
a) *Software engineering* is a field of *engineering*, for designing and writing programs for computers or other electronic devices. A *software engineer*, or programmer, writes *software* (or changes existing *software*) and compiles *software* using methods that improve it. Better quality *software* is easier to use.

According to IEEE's definition it is an application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.

b) A software engineering process is the model chosen for managing the creation of software from initial customer inception to the release of the finished product. The chosen process usually involves techniques such as

1. Analysis,
2. Design,
3. Coding,
4. Testing and
5. Maintenance

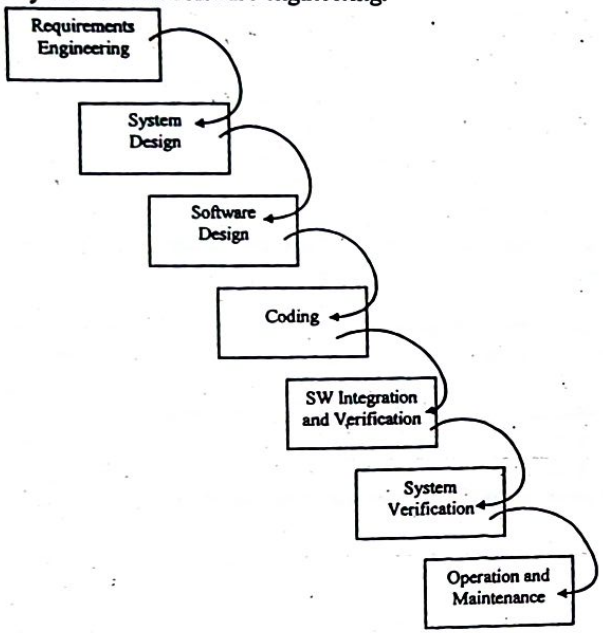
Several different process models exist and vary mainly in the frequency, application and implementation of the above techniques, for example, different process models use different analysis techniques, other models attempt to implement the solution to a problem in one big-bang approach, while others adopt an iterative approach whereby successively larger and more complete versions of the software are built with each iteration of the process model. E.g. the various phases of what is probably the oldest software development process in existence, namely the classic life-cycle paradigm, sometimes called the "waterfall model". This paradigm implies a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and maintenance. Modeled after the conventional engineering cycle, the life-cycle paradigm encompasses the above activities.



c) 1st Part:

The Waterfall Model

The waterfall model is believed to have been the first process model which was introduced and widely followed in software engineering.



The phases of "The Waterfall Model" are:

Requirement Analysis & Definition: All requirements of the system which has to be developed are collected in this step. Like in other process models requirements are split up in functional requirements and constraints which the system has to fulfill.

Requirements have to be collected by analyzing the needs of the end user(s) and checking them for validity and the possibility to implement them. The aim is to generate a Requirements Specification Document which is used as an input for the next phase of the model.

System Design: The system has to be properly designed before any implementation is started. This involves an architectural design which defines and describes the main blocks and components of the system, their interfaces and interactions. By this the needed hardware is defined and the software is split up in its components. E.g. this involves the definition or selection of a computer platform, an operating system, other peripheral hardware, etc. The software components have to be defined to meet the end user requirements and to meet the need of possible scalability of the system. The aim of this phase is to generate a System Architecture Document this serves as an input for the software design phase of the development, but also as an input for hardware design or selection activities. Usually in this phase various documents are generated, one for each discipline, so that the software usually will receive a software architecture document.

Software Design: Based on the system architecture which defines the main software blocks the software design will break them further down into code modules. The interfaces and interactions of the modules are described, as well as their functional contents. All necessary system states like startup, shutdown, error conditions and diagnostic modes have to be considered and the activity and behavior of the software has to be defined. The output of this phase is a Software Design Document which is the base of the following implementation work.

Coding: Based on the software design document the work is aiming to set up the defined modules or units and actual coding is started. The system is first developed in smaller portions called units. They are able to stand alone from an functional aspect and are integrated later on to form the complete software package.

Software Integration & Verification: Each unit is developed independently and can be tested for its functionality. This is the so called Unit Testing. It [illegible]s if the modules or units to check if they meet their specifications. This involves functional tests at the interfaces of the modules, but also more detailed tests which consider the inner structure of the software modules. During integration the units which are developed and tested for their functionalities are brought together. The modules are integrated into a complete system and tested to check if all modules cooperate as expected.

System Verification: After successfully integration including the related tests the complete system has to be tested against its initial requirements. This will include the original hardware and environment, whereas the previous integration and testing phase may still be performed in a different environment or on a test bench.

Operation & Maintenance: The system is handed over to the customer and will be used for the first time by him. Naturally the customer will check if his requirements were implemented as expected but he will also validate if the correct requirements have been set up in the beginning. In case there are changes necessary it has to be fixed to make the system usable or to make it comply to the customer wishes. In most of the "Waterfall model" descriptions this phase is extended to a never ending phase of "Operations & Maintenance". All the problems which did not arise during the previous phases will be solved in this last phase.

2nd Part:

Advantages

- Simple and easy to 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.

Disadvantages

- Adjusting scope during the life cycle can kill a project
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Poor model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Poor model where requirements are at a moderate to high risk of changing.

6. a) What is the Heuristic Project Estimation technique? [WBUT 2017]

Answer:

Refer to Question No. 10(a) [Heuristic Techniques], Long Answer Type Questions.

b) A Project size of 200 LOC is to be developed. Software development team has average experience on similar type of project. The project schedule is not very tight. Calculate Effort, Time of Development, Average Staff size and Productivity of the Project. [WBUT 2017]

Answer:

i) Effort= $2.4(KLOC)^{1.05} = 2.4(200)^{1.05} = 625.5942$

Tdev=$2.5(Effort)^{0.38}$=$2.5(625.5942)^{0.38}$=28.88 months

Average Staff Size=Effort/TimeSchedule= 21.66

Productivity level=[KLOC/EFFORT] DSI/MM=200/625.5942

=0.3197 DSI/MM

ii) Effort= $3.0(KLOC)^{1.12} = 3.0(200)^{1.12} = 1133.1172$

Tdev=$2.5(Effort)^{0.35}$=$2.5(1133.1172)^{0.35}$=29.30 months

Average Staff Size=Effort/TimeSchedule= 38.67

Productivity level= [KLOC/EFFORT] DSI/MM=200/1133.1172

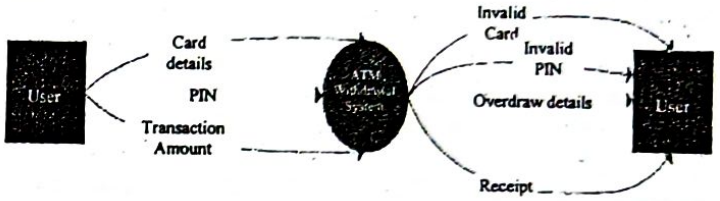
= 0.1765 DSI/MM

7. a) Draw a DFD that depicts and ATM system (only withdrawal) mentioning suitable Assumptions. Now build a structure chart.

b) An embedded system of size 400 KLOC has to develop. Project manager has a choice of hiring from two pools of developers: very highly capable with very little experience (AEXP very high) or developers of low quality (LEXP very low) but a lot of Experience. Find the values of EAF, Effort & Development Time. [WBUT 2017]

Answer:

a)



Context Diagram - ATM (Cash Withdrawal) System - Data Flow Diagram Template

Values of EAF, Effort & Development

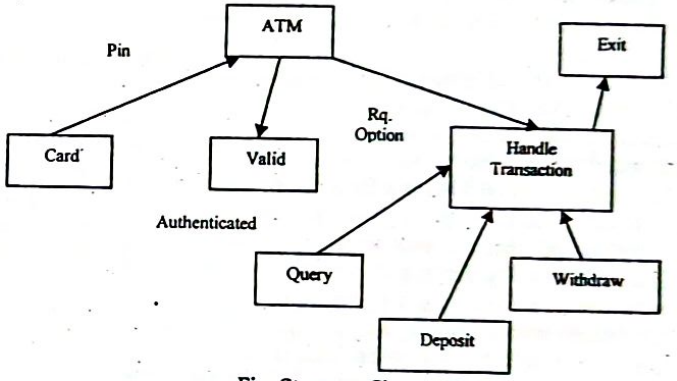


Fig: Structure Chart - ATM

b) This is the case of embedded mode and model is intermediate COCOMO. Hence,

$E = a_i(KLOC)^{b_i} * EAF$

$D = c_i(E)^{d_i}$

$= 2.8(400)^{1.20} = 3712$ PM

Case I: Developers are very highly capable with very little experience in the programming being used.

EAF = 0.82 x 1.14 = 0.9348
E = 3712 x .9348 = 3470 PM
D = 2.5 $(3470)^{0.32}$ = 33.9 M

Case II: Developers are of low quality but lot of experience with the programming language being used.

EAF = 1.29 x 0.95 = 1.22
E = 3712 x 1.22 = 4528 PM
D = 2.5 $(4528)^{0.32}$ = 36.9 M

8. What is cost benefit analysis? What are the common techniques for cost benefit analysis? Develop a set of functional and non-functional requirements for a new software project. **[WBUT 2018]**

Answer:

1st Part:

A benefit-cost ratio analysis (BCR) is an indicator used in cost-benefit analysis to show the relationship between the relative costs and benefits of a proposed project. Benefit-cost ratios (BCRs) are most often used in capital budgeting to analyze the overall value for money of undertaking a new project. However, the cost-benefit analyses for large projects can be hard to get right, because there are so many assumptions and uncertainties that are hard to quantify.

2nd Part:

The common techniques for cost benefit analysis are- cost-utility analysis, risk-benefit analysis, economic impact analysis, social return on investment (SROI) analysis.

- **Cost-utility analysis:** Cost–utility analysis (CUA) is a form of financial analysis used to guide procurement decisions. The most common and well-known application of this analysis is in pharmacoeconomics, especially health technology assessment (HTA). Cost-utility analysis allows comparison across different health programs and policies by using a common unit of measure (money/QALYs gained). Cost-utility analysis provides a more complete analysis of total benefits than simple cost–benefit analysis does. This is because Cost-utility analysis takes into account the quality of life that an individual has, while Cost-benefit analysis does not.
- **Risk-benefit ratio:** A risk–benefit ratio is the ratio of the risk of an action to its potential benefits. Risk–benefit analysis is analysis that seeks to quantify the risk and benefits and hence their ratio. Analyzing a risk can be heavily dependent on the human factor. A certain level of risk in our lives is accepted as necessary to achieve certain benefits.
- **Economical impact analysis:** An economic impact analysis (EIA) examines the effect of an event on the economy in a specified area, ranging from a single neighborhood to the entire globe. It usually measures changes in business

revenue, business profits, personal wages, and/or jobs. The economic event analyzed can include implementation of a new policy or project, or may simply be the presence of a business or organization. An economic impact analysis is commonly conducted when there is public concern about the potential impacts of a proposed project or policy. An economic impact analysis typically measures or estimates the change in economic activity between two scenarios, one assuming the economic event occurs, and one assuming it does not occur (which is referred to as the counterfactual case). This can be accomplished either before or after the event.

- **Social return on investment:** Social return on investment (SROI) is a principles-based method for measuring extra-financial value (such as environmental or social value not currently reflected or involved in conventional financial accounts). It can be used by any entity to evaluate impact on stakeholders, identify ways to improve performance, and enhance the performance of investments. The SROI method as it has been standardized by the Social Value UK provides a consistent quantitative approach to understanding and managing the impacts of a project, business, organisation, fund or policy. It accounts for stakeholders' views of impact, and puts financial 'proxy' values on all those impacts identified by stakeholders which do not typically have market values. The aim is to include the values of people that are often excluded from markets in the same terms as used in markets, that is money, in order to give people a voice in resource allocation decisions.

3rd Part:

Functional Requirements define what the system needs to do, Functional requirements are what the business users expect the software to 'do'; what tasks they wish it to perform. (e.g. write paychecks, calculate launch date for lunar orbit, do hours to gross calculation for movie studios, etc.) .

Whereas non Functional Requirements describe how the system will do it. The functional requirement is to produce a pay slip for an employee, the non-functional requirement is to print a pay slip per employee (up to, say, 10,000) in, say, 48 hours elapsed. The non functional requirement may be one of the Critical Success Factors, it may even be the prime driver for the development, but it is still non-functional. Non functional requirements have also been called the 'ilities' because they are most simply expressed like this:

6. usability
7. reliability
8. interoperability
9. scalability
10. security

Functional Requirements are either met or not met. Non-functional requirements tend to be things that you can measure.

Example: The **Functional requirements** for the **Librarian** are:

Insert book:
This action is done to add new book to library book collection

Delete / modify book: This event is to delete an existing book or modify its information.

Delete member: Admin can delete a member due to some specific rules.

Return book: Admin should confirm the return of books borrowed by users.

Non-functional Requirements for the Librarian are:

Error handling: Library Management System shall handle expected and non-expected errors in ways that prevent loss in information and long downtime period.

Performance Requirements: The system shall accommodate high number of books and users without any fault.

Safety Requirements: System use shall not cause any harm to human users

Security Requirements: System will use secured database.

9. a) Consider a project with 950 lines of code which has the following distribution in terms of days for each effort:

Phase	Programmer Days
Requirements	20
Design	10
Implementation	10
Testing	15
Documentation	10

Calculate the productivity given the line of code and number of programmer days.

b) Calculate COCOMO effort, development time and productivity for an organic project that is estimated to have 39,800 lines of code. Assume values of constant a = 2.4 and b = 1.05. **[WBUT 2018]**

Answer:

a) Total Efforts = (20+10+10+15+10) = 65

Productivity = LOC/ Total Efforts

= 950/65

= 14.6 LOC/programmer's day

b) For Organic model,

$$\text{Lines of codes} = 39800 = \frac{39800}{1000} = 39.8 KLOC$$

$a = 2.4; b = 1.05$

$$\text{The COCOMO effect (E)} = 2.4(39.8)^{1.05} PM = 10.235 PM$$

$$T_{dev} = 2.5(10.235)^{0.38} = 6.050 = 6 Months (Approx)$$

$$\text{Therefore, Productivity} = \frac{KLOC}{Effect} = \frac{39.8}{10.235} DSI / MM = 3.888$$

10. Write short notes on the following:

a) Software Project Plan **[WBUT 2013]**
b) Software Configuration Management **[WBUT 2014, 2015, 2018]**
c) Risk management **[WBUT 2015]**
d) CASE tools **[WBUT 2015, 2017, 2018]**
e) Waterfall Model **[WBUT 2016, 2018]**
f) COCOMO Model **[WBUT 2016]**
g) DFD **[WBUT 2016]**
h) Team Structure of an Organization **[WBUT 2017]**
I) Risk Identification & Assessment **[WBUT 2017]**
j) RAD model **[MODEL QUESTION]**

Answer:

a) Software Project Plan:

Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated include: project size, effort required to develop the software, project duration, and cost. These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling. There are three broad categories of estimation techniques:

1. Empirical Estimation Techniques:

Empirical estimation techniques are based on making an educated guess of the project parameter: While using this technique, prior experience with development of similar products is helpful. Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years.

2. Heuristic Techniques:

Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression.

3. Analytical Estimation Techniques:

Analytical estimation techniques derive the required results starting with basic assumption regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. Halstead's software science is an example of an analytical technique. Halstead's software science can be used to derive some interesting results starting with a few simple assumptions.

b) Software Configuration Management:

The output of the software process is information that may be divided into three broad categories:

1. Computer programs (both source level and executable forms).
2. Documents that describe the computer programs (targeted at both technical papers and users).
3. Data (contained within the program or external to it).

The items that comprise all information produced as part of the software process are collectively called a **software configuration.** As the software process progresses, the number of software configuration (SCIs) grows rapidly. A System Specification spawns a Software Project Plan and Software Requirements Specification (as well as hardware related documents). These in turn spawn other documents to create a hierarchy of information. There are four fundamental sources of change:

1. New business or market conditions dictate changes in product requirements or business rules.
2. New customer needs demand modification of data produced by systems, functionality delivered by products, or services deliver computer-based system.
3. Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
4. Budgetary or scheduling constraints cause a redefinition of the system or product.

Software configuration management is a set of activities that have been developed to manage change throughout the life-cycle of computer software.
Customers want to modify requirements. Developers want to modify the technical approach. Managers want to modify the project strategy. Why all this modification? The answer is really quite simple. As time passes, all constituencies know more (about what they need, which approach would be best, how to get it done and still make money). This additional knowledge is the driving force behind most changes.

c) Risk management:
Refer to Question No. 4(b) of Long Answer Type Questions.

Risk control:
It is the process of managing risks to achieve the desired outcomes. Risk management planning produces a plan for dealing with each significant risk. It is useful to record decisions in the plan, so that both customer and developer can review how problems are to be avoided, as well as how they are to be handled when they arise. We should also monitor the project as development progresses, periodically reevaluating the risks, their probability, and likely impact. Risk resolution is the execution of the plans for dealing with each risk. The risk management practice is to identify the nature and status of the risk, write them down, communicate them to concerned people and authority including insurance consultants over the duration of the project.

d) CASE tools:
CASE (Computer Aided Software Engineering) tools capture the data items appearing in a DFD automatically to generate the data dictionary. CASE tools support queries about definition and usage of data items.
For example, queries may be made to find which data item affects which processes a process affects which data items the definition and usage of specific data items, etc. Query handling is facilitated if data dictionary is stored in a relational database management system (RDBMS). Several CASE tools are available:

- To support structured analysis and design
- To maintain the data dictionary
- To check whether DFDs are balanced or not

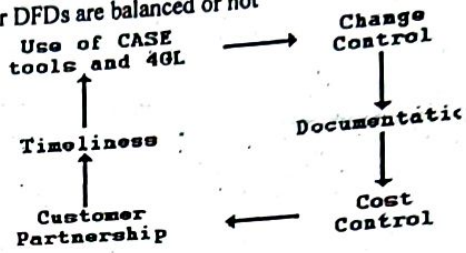


Fig: CASE Tools

The tasks like Systems Analysis and drawings of Data Flow Diagrams are falling under the category of **Lower Case.** However, **Upper Case** focuses on tools found in different CASE products to help in the analysis and design process. Some of the **Upper Case tools** are **screen generators, report generators etc.**

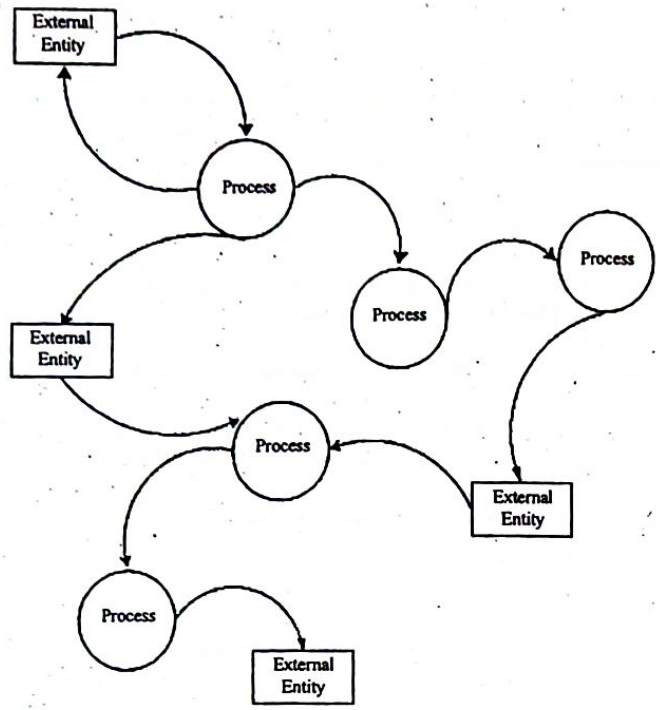
e) Waterfall Model: ***Refer to Question No. 5(c) of Long Answer Type Questions.***

f) COCOMO Model: ***Refer to Question No. 2 of Long Answer Type Questions.***

g) DFD:
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored (all of which are shown on a DFD).



Physical data flow diagrams show how the system operates or how the new system will be implemented.

- Physical data flow diagrams include
- Clarifying which processes are manual and which are automated
- Describing processes in greater detail
- Sequencing processes in the order they must be executed
- Temporary data stores and transaction files
- Specifying actual document and file names
- Controls to ensure accuracy and completeness
- Describes processes for adding, reading, changing, and deleting records
- CRUD is an acronym for Create, Read, Update, Delete
- A CRUD matrix shows which programs or processes add, read, update, or delete master file records.

Logical data flow diagrams show how the business operates. It shows the processes that would exist regardless of the type of system implemented

- The progression of creating data flow diagrams is
- Create a logical DFD of the current system

- Next add all the data and processes not in the current system which must be present in the new system
- Finally derive the physical data flow diagram for the new system
- Advantages of logical DFDs are
- Better communication with users
- More stable systems, since the design is based on a business framework.
- Increased understanding of the business by analysts
- The system will have increased flexibility and be easier to maintain
- Elimination of redundancy

Disadvantages of data flow diagram

DFD is likely to take many alterations before agreement with the user
Physical considerations are usually left out
It is difficult to understand because it ambiguous to the user who have little or no knowledge.

h) Personnel required for a program or a project is a practice of finding, evaluating, and establishing a working environment and fires them when they are no longer needed. Staffing involves finding people, who may be hired or already working for the company (organization) or may be working for competing companies. Staffing management plan is a part of software development.

- It may not be possible to appoint the ideal people to work on a project
- Project budget may not allow for the use of highly-paid staff
- Staff with the appropriate experience may not be available

An organisation may wish to develop employee skills on a software project. Managers have to work within these constraints especially when there are shortages of trained staff. There is a risk if right kind of people are not available in a project, it makes a big difference whether a project is staffed with capable engineers. If key positions are filled-up with inappropriate people, the project runs the risk of delaying deliveries or producing poor-quality products, or both.

Project Team

Staffing management plan is a part of software development.

- It may not be possible to appoint the ideal people to work on a project
- Project budget may not allow for the use of highly-paid staff;
- Staff with the appropriate experience may not be available;

An organisation may wish to develop employee skills on a software project. Managers have to work within these constraints especially when there are shortages of trained staff. There is a risk if right kind of people is not available in a project; it makes a big difference whether a project is staffed with capable engineers. If key positions are filled-

up with inappropriate people, the project runs the risk of delaying deliveries or producing poor-quality products, or both.

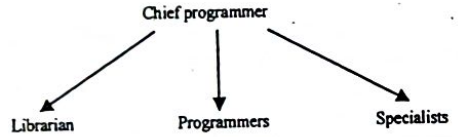
Number of staff required in a project depends on an ideal logical view of a software engineering environment, which covers:

- The infrastructure is the heart of the environment: all connections among different components take place through it.
- Individual tools constitute a layer that uses the services provided by the infrastructure.
- Methods and methodologies—the external layers of the structure—provide guidance on the use of tools. They enforce a predefined strategy of using the tools according to the principles that underlie the method or the methodology.

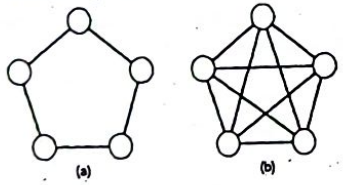
Finally, the environment must support group work: designers do not work in isolation, nor can they interact only through the database. Much critical activity in the software production process is of a "social type" (brainstorming sessions, walk-through, code inspections, etc.) not presently supported by the available commercial technology.

Some of the software organizations are:

- Centralized-Control Team Organization

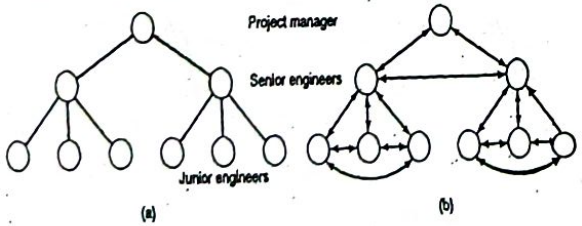


- Decentralized-Control Team Organization



(a) Management structure. (b) Patterns of communication.

- Mixed-Control Team Organization



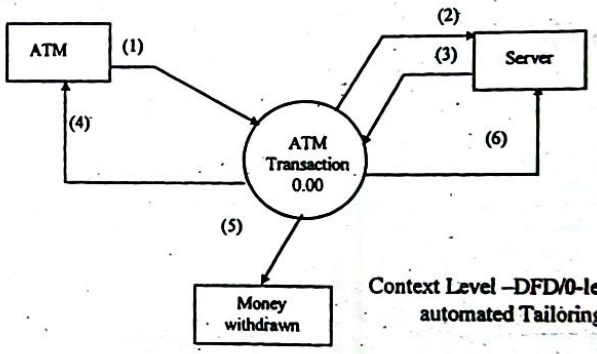
Mixed-controlorganizations.
(a) Management structure. (b) Patterns of communication.
An Assessment of Team Organizations

i) Risk Identification & Assessment:
Refer to Question No. 4 (b) of Long Answer Type Questions.

j) RAD model:
The Rapid Application Development Model is a property of IBM. User involvement is essential at every stage of the model. At the first step of the model building a prototype is built. RAD-based methodologies adjust the SDLC phases to get some part of system developed quickly and into the hands of the users. Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as CASE (computer-aided software engineering) tools. One possible subtle problem with RAD-based methodologies is managing user expectations.
According to the functional requirements the model is separated into four stages:

- Requirement Planning
- User Description
- Construction Phase
- Cut over Phase

11. Draw and explain the DFD for ATM transaction system. [MODEL QUESTION]
Answer:



Context Level –DFD/0-level DFD for an automated Tailoring machine

(1) Customer at the ATM key in the public and private key to access the system
(2) Against the server the private and the public key is checked
(3,4) Server authenticate/ reject the user
(5) Money Transaction performed
(6) Balance updated

SOFTWARE DESIGN & UML

Multiple Choice Type Questions

1. To allocate resource to activities we use [WBUT 2014]
   a) PERT chart b) Gnatt chart
   c) Network Diagram d) all of these
Answer: (d)

2. To achieve a good design, modules should have – [WBUT 2014]
   a) weak cohesion low coupling b) weak cohesion high coupling
   c) strong cohesion low coupling d) strong cohesion high coupling
Answer: (c)

3. If data from one module is used to direct the order of execution in another, then the coupling is known as – [WBUT 2014]
   a) Stamp Coupling b) Data Coupling
   c) Control Coupling d) Content Coupling
Answer: (c)

4. Cardinality in an ER Diagram refers to [WBUT 2014]
   a) number of attributes in an entity b) the order of co-related entities
   c) the number of sub-entities d) degree of a relationship
Answer: (b)

5. The bet type of cohesion is [WBUT 2015]
   a) coincidental b) logical c) informational d) functional
Answer: (d)

6. A physical DFD specifies [WBUT 2015]
   a) what processes will be used
   b) who generates data and who processes it
   c) what each person in an organization does
   d) none of these
Answer: (d)

7. The most creative and challenging phase of system life cycle is [WBUT 2017]
   a) Design b) Feasibility study
   c) Maintenance d) none of these
Answer: (a)

8. The database design activity deals with the design of [WBUT 2017]
   a) Logical database b) Physical database
   c) both (a) and (b) d) none of these
Answer: (c)

9. Coupling is a measure of [WBUT 2017]
   a) Relative functional strength b) Interdependence among module
   c) both (a) and (b) d) none of these
Answer: (c)

10. When the two bubbles are interconnected directly, it is referred as [WBUT 2018]
   a) serial DFD b) direct DFD
   c) synchronous d) balanced DFD
Answer: (c)

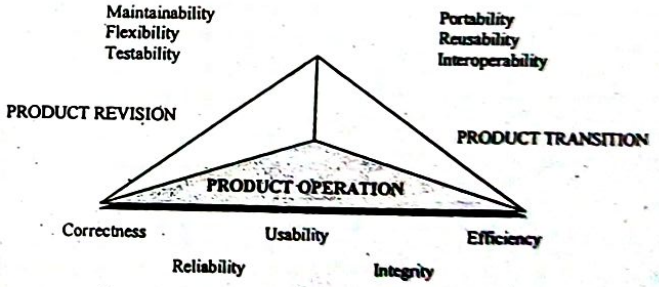
Short Answer Type Questions

1. a) Define software quality. [WBUT 2013, 2015]
   b) Briefly explain McCall's quality factors. [WBUT 2013]
Answer:

a) Software quality is the degree of conformance to explicit or implicit requirements and expectations.

b) The factors that affect software quality can be categorized in two broad groups:
Factors that can be directly measured (e.g., defects uncovered during testing).
Factors that can be measured only indirectly (e.g., usability or maintainability).
In each case measurement should occur. We must compare the software (programs, data, documents) to some datum and arrive at an indication of quality.
McCall, Richards, and Walters propose a useful categorization of factors that affect software quality. These software quality factors, focus on three important aspects of a software product:

1. Operational characteristics,
2. Ability to undergo change,
3. Adaptability to new environments.



Referring to the factors noted in Figure above McCall and his colleagues provide the following descriptions:

1. *Correctness:* The extent to which a program satisfies its specification and fulfills the customer's mission objectives.

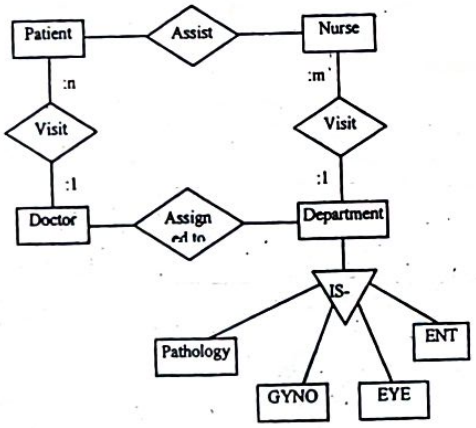
2. ***Reliability:*** The extent to which a program can be expected to perform its intended function with required precision. (It should be noted that other, more complete definitions of reliability have been proposed.
3. ***Efficiency:*** The amount of computing resources and code required by a program to perform its function.
4. ***Integrity:*** The extent to which access to software or data by unauthorized persons can be controlled.
5. ***Usability:*** The effort required learning, operating, preparing input for, and interpreting output of a program.
6. ***Maintainability:*** The effort required to locate and fix an error in a program. This is a very limited definition.
7. ***Flexibility:*** The effort required to modify an operational program.
8. ***Testability:*** The effort required to test a program to ensure that it performs its intended function.
9. ***Portability:*** The effort required to transfer the program from one hardware and/or soft ware system environment to another.
10. ***Reusability:*** The extent to which a program [parts of a program] can be reused in other applications — related to the packaging and scope of the functions that the program performs.
11. ***Interoperability:*** The effort required to couple one system to another.

2. Draw an ER diagram for Hospital Management System showing cardinalities, strong and weak entities, derived attributes, primary key etc. [WBUT 2014]

OR,

Draw the E-R diagram for a hospital management. [WBUT 2016]

Answer:



DOCTOR{Doctor-id, name, specialization, sex, age, contactno}
PATIENT{Patient-ID, name, diesis, doctor-id(FK), Nurse-ID(FK)}

NURSE{Nurse-ID, name,department, Specialization,contactno)
DEPARTMENT{Department-ID, name, Doctor-id, Nurse-ID,Type}
Note: There is no weak entity.

3. What are Cohesion and Coupling? Mention different kinds of cohesion. [WBUT 2015]

Answer:

Refer to Question No. 13(b) of Long Answer Type Questions.

4. What is Use Case diagram? Draw the Use Case diagram of Hospital Management System. [WBUT 2015]

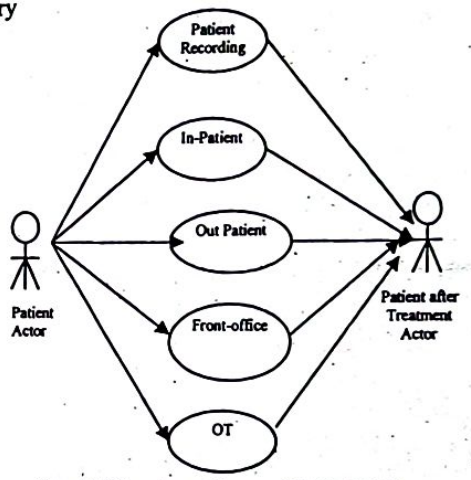
Answer:

1st Part:

Use Case Diagram displays the relationship among actors and use cases.

2nd Part:

Out-PatentDispensery



Hospital Management System USE CASE Diagram

5. It is estimated that there will be 70 errors in a software. During testing 25 errors have been experienced. Calculate the failure intensity with a given value of $\phi = 0.03$ using Jelenski Moranda model. What will be the failure intensity after experiencing 50 errors? [WBUT 2015]

Answer:

The Jelinski_Moranda model is one of the most primitive reliability model. The functional representation of the failure intensity can be expressed in the form
Rate of failure(t) = Φ(N-i+1), where time interval of failure is assumed to be same
N= Total number of errors present
i= Number of errors found at time intervals

The set of observations for our case is:

N=70 errors

I=20 failure

Φ=0.03 (given)

Rate of failure(t)=0.03(70-25+1)=0.03x46 =1.38 failure/cpu Hr.

After 50 failures

Rate of failure(t)=0.03(70-50+1)=0.03x19 = .57 failure/cpu Hr

Hence the rate of failures is decreasing with number of failures

6. What is meant by cohesion? How should software be designed considering cohesion? What is the difference between cohesion and coupling? [WBUT 2017]

Answer:

Refer to Question No. 13(b) of Long Answer Type Questions.

7. What are the metrics for estimation of software? [WBUT 2018]

Answer:

Software metric is a measure of software characteristics which are quantifiable or countable. Software metrics are important for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

The metrics for estimation of software can be classified into three categories: product metrics, process **metrics**, and project **metrics**. Product **metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level. Process **metrics** can be used to improve **software** development and maintenance.

8. Specify the general principles of user interface design. [WBUT 2018]

Answer:

The principles of user interface design are intended to improve the quality of user interface design. These principles are:

(i) The structure principle: Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

(ii) The simplicity principle: The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

(iii) The visibility principle: The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.

(iv) The feedback principle: The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

(v) The tolerance principle: The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonable.

(vi) The reuse principle: The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

9. What are Transformation analysis and Transaction analysis? [MODEL QUESTION]

Answer:

Transform Analysis

Transform analysis identifies the primary functional components (modules) and the high level input and outputs for these components. The first step in transform analysis is to divide the DFD into three types of parts: Input, Logical processing, Output. The input portion in the DFD includes processes that transform input data from physical to logical form. Each input is called an afferent branch. The output of a DFD transforms output data from logical form to physical form. Each output portion is called an efferent branch. The remaining portion of a DFD is referred as central transform. In the next step of transform analysis, the structure chart is derived by drawing one functional component for the central transform and the afferent (inward) and efferent (outward) branches.

Transaction Analysis

A transaction allows the user to perform some meaningful piece of work. Transaction analysis is an alternative to transform analysis and is useful while designing transaction-processing programs. In a transaction driven system, one of several possible paths through the DFD is traversed depending upon the input data item. This is in contrast to a transform-centered system, which is characterized by identical processing steps for each data item.

Long Answer Type Questions

1. a) Briefly describe different user interface elements. [WBUT 2013]

b) What are the methodology for dialog design?

Answer:

a) When designing your interface, try to be consistent and predictable in your choice of interface elements. Whether they are aware of it or not, users have become familiar with elements acting in a certain way, so choosing to adopt those elements when appropriate will help with task completion, efficiency, and satisfaction.

Interface elements include but are not limited to:
Input Controls: checkboxes, radio buttons, dropdown lists, list boxes, buttons, toggles, text fields, date field
Navigational Components: breadcrumb, slider, search field, pagination, slider, tags, icons
Informational Components: tooltips, icons, progress bar, notifications, message boxes, modal windows
Containers: accordion

b) Dialogue is a literary and theatrical form consisting of a written or spoken conversational exchange between two or more people.
Dialogue is used in classrooms, community gatherings, federal agencies, and other public addressing places to enable people, usually in small groups to share their perspectives and experiences about difficult issues. It is used to help people resolve long-standing conflicts and to build deeper understanding of contentious issues. Dialogue is about understanding and learning. Dialogue dispels stereotypes, builds trust, and enables people to be open to perspectives that are very different from their own.
Dialogic relations have a specific nature: they can be reduced neither to the purely logical (even if dialectical) nor to the purely linguistic (compositional-syntactic). They are possible only between complete utterances of various speaking subjects... Where there is no word and no language, there can be no dialogic relations; they cannot exist among objects or logical quantities (concepts, judgments, and so forth). Dialogic relations presuppose a language, but they do not reside within the system of language. They are impossible among elements of a language.
Today, dialogue is used in classrooms, community centers, corporations, federal agencies, and other settings to enable people, usually in small groups, to share their perspectives and experiences about difficult issues. It is used to help people resolve long-standing conflicts and to build deeper understanding of contentious issues. Dialogue is not about judging, weighing, or making decisions, but about understanding and learning. Dialogue dispels stereotypes, builds trust, and enables people to be open to perspectives that are very different from their own.

Structured dialogue: Structured dialogue represents a class of dialogue practices developed as a means of orienting the dialogic discourse toward problem understanding and consensual action. Whereas most traditional dialogue practices are unstructured or semi-structured, such conversational modes have been observed as insufficient for the coordination of multiple perspectives in a problem area. A disciplined form of dialogue, where participants agree to follow a framework or facilitation, enables groups to address complex problems shared in common.

Process of designing a dialogue
Designing Dialogue

- A Dialogue is the sequence in which information is displayed to and obtained from a user.

- As designer the role is to select the most appropriate interaction methods and devices and to define the condition under which information is displayed to and obtained from a user
- Three major step to deigning dialogue:
  1. Designing the dialogue sequence
  2. Building a prototype
  3. Assessing usability

For a dialogue to have high usability, it must be consistent in form, function, and style.
Designing the dialogue sequence for a typical dialogue between user and the customer information system for obtaining this information might proceed:
1. Request to view individual customer information
2. Specify the customer of interest
3. Select the year to date transaction summary display
4. Review customer information
5. Leave system

- Dialogue diagramming is a formal method for designing and representing human computer dialogues using box and line diagram
- The three section of the box are used:
  – Top: contains a unique display ref. number
  – Middle: contains the name or description of the display
  – Bottom: contain display ref. number that can be accessed from the current display

Designing Interfaces and Dialogues in Graphical Environments an effective GUI designer:
1. Become an expert user of the GUI environment
2. Understand the available resources and how they can be used.

- Common properties of Windows and forms in GUI environment that can be active or inactive:
  - Modality
  - Resizable
  - Movable
  - Minimize
  - Maximize
  - System menu

Common Errors when designing the interface and dialogues of website.

- Opening new browser window
- Breaking or slowing down the back button
- Complex URLs
- Orphan pages
- Scrolling navigation pages
- Lack of navigation support
- Hidden links

- Links that don't provide enough information
- Buttons that provide no click feedback.

2. a) Draw the context-diagram and level-1 diagram along with a data dictionary of an examination system for processing the marks of the students. The system after processing the marks of students generates a report card for each individual student. The system also generates a consolidated report and sends it to the controller of examination. [WBUT 2013]

OR,

Draw the context diagram and Level 1 and Level 2 DFD for the following correspondence course system. A college offers correspondence courses to students. Each course lasts 20 weeks and is based on a weekly study module and progress test. At the end of the course students sit at an invigilated examination. The college Registrar deals with enquiries and applications, and students applying who have sufficient qualifications are asked to register by completing and submitting an application form. After approval by the Academic Director, the application form is returned to the Registrar who creates a student file. The Accounts department receives the application form and using information from the student file creates an invoice that is sent to the student. Payments made are registered on the invoice file. The first batch of student material and tests is issued from the library only to students who have paid fees (this information is taken from the invoice file). [WBUT 2018]

b) Also draw the Use Case diagram of the above mentioned scenario. [WBUT 2013]

c) What are the propositions of Putnam's model? [WBUT 2013, 2018]

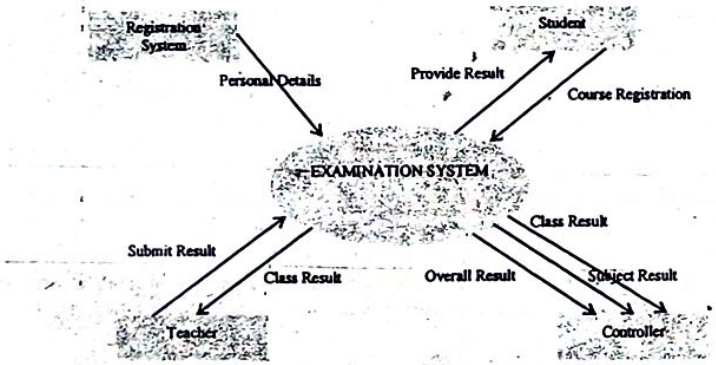
d) Discuss the different stages of 'Capability Maturity Model'. [WBUT 2013]

OR,

Explain in detail the Capability Maturity Model (CMM). [WBUT 2016]

Answer:

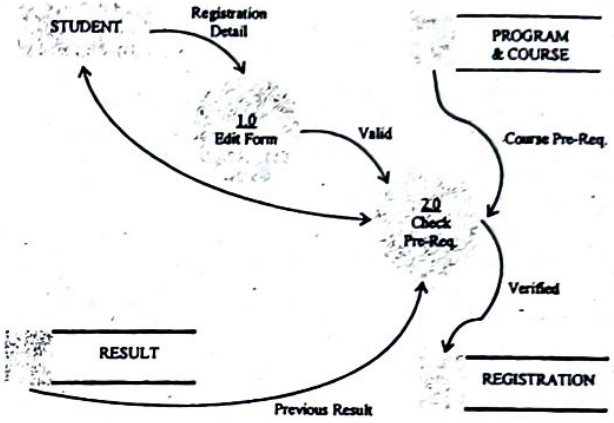
a)



Level 0 Diagram

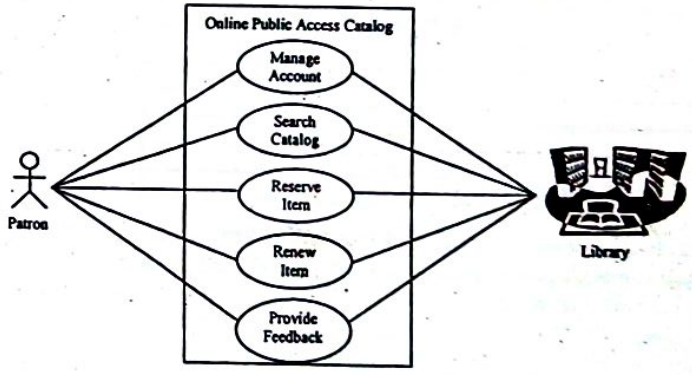
The three major modules which have been identified are given below our level 0 diagram will be based on these three modules and will elaborate and describe each of the modules in details.

- Subject registration
- Result submission
- Result calculation



The first module identified in the system is the Registration of the students for the system. As the DFD show a student applies for registration along-with certain registration information which is required by the system, Process 1.0 of the system checks the validity of information in the form, if the Registration form is found to be valid the information in the form is passed onto the second process where the validity of registration is determined by checking certain prerequisites for the courses to which student wishes to be enrolled and the database will be updated accordingly. During this process the result of the students is also checked for the previous semester or previously studied subject to confirm whether the student has passed a certain pre requisite subject before he can attempt to enroll for a second course which is based on that prerequisite.

b)



Draw a USE Case diagram of a Library Management System (LMS), which can search library catalog online to locate various resources - books, periodicals, audio and visual materials, or other items under control of the library. Patrons may reserve or renew item, provide feedback, and manage their account.

c) The Putnam model is an experimental software effort assessment model. A group of empirical models work by collecting software project data containing 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. 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:

- There is a highly non-linear relationship between chronological time to complete a project and human effort applied.
- The number of delivered lines of code L is related to effort and development time by the equation:
  $L = P \times E^{1/3} t^{4/3}$
- E is development effort in person-months.
  P is a productivity parameter that reflects that result in high quality (typically 2,000-12,000), t is the project duration in calendar months.
- Rearranged to solve for development effort:
  $E = L^3/(P^3 t^3)$

E is effort expended (in person-years) over the entire project life-cycle.
L is lines of code (source statements).
P is a productivity parameter that reflects that result in high quality. (Typically 2,000-12,000). t is the development time in years.

d) The CMM model was developed by SET (Software Engineering Institute) of Carnegie-Mellon University in 1986. The model is a strategy for improving the software process, irrespective of the actual life cycle model used, It is used to judge the maturity of the software processes of an organization and to identify the key practices. The CMM is organized into **five maturity** levels summarized as follows:

Initial	Characterization
Repeatable	Basic Project Management
Deferred	Process Definition
Managed	Process Measured
Optimizing	Process Control

Except for level 1, each maturity level is decomposed into several key process areas that indicate the areas (KPAs) an org. should focus on to improve its software process. There are no key process areas for *level 1*.

The key process areas at *level 2* focus on the software project's concerns related to establishing basic project management controls, as summarized below:

1. Requirements Management (RM)
2. Software Project Planning (PP)
3. software project Tracing and Oversight (PT).
4. Software Subcontract Management (SM)
5. Software Quality Assurance (QA)
6. Software Configuration Management (CM)

The key process areas at *level 3* address both project and organizational issues as summarized below:

1. Organization Process Focus (PF)
2. Organization Process Definition (PD)
3. Training Program (TP)
4. Integrated Software Management (IM)
5. Software Product Engineering (PE)
6. Inter group Coordination (IC)
7. Peer Reviews (PR)

The key process areas of *level 4* focuses on establishing a quantitative understanding of both the software process and the software work products as summarized below:

1. Quantitative Process Control Management (QP)
2. Software Quality Management (QM)

The key process areas of *level 5* cover the issues that both the organization and the projects as summarized below:

1. Defect Prevention (DP)
2. Technology Change Management (TM)

Process Change Management (PC)

3. a) Explain "Use Case" diagram. What are the essential criteria for ideal use case diagram? What are the "extends" and "includes" constructs in use case diagram? Draw a use case diagram for Nursing Home functionality where examples of actors are Patient, Doctor, Reception Staff, Billing Staff and Administrator etc.
b) Explain Sequence and Activity diagram with example. **[WBUT 2014]**

Answer:

a) 1st Part:

To model a system the most important aspect is to capture the dynamic behavior of the system. The purpose of use case diagram is to capture the dynamic aspect of a system through internal and/or external factors for making the interaction. These internal and external agents are known as actors. Use case diagrams consists of actors and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. To model the entire system a number of use case diagrams are used. A Use Case describes from the

point of view of the actors. A group of activities in a system produces a concrete, tangible result.

A use case is a sequence of actions that provide a measurable value to an actor. An essential use-case can be defined as follows:

It a generalized use casa simplified, abstract, that captures the intentions of a user in a technology independent manner without any indication of implementation.

It is a structured narrative, expressed in the language of the application, comprising a simplified, generalized, abstract, description of a task with interactions.

It is complete, meaningful, and well designed from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction.

2nd Part:

Includes

In one form of interaction, a given use case may *include* another. "Include is a Directed Relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case

An include relationship is depicted with a directed arrow having a dotted line. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow. The stereotype "<<include>>" identifies the relationship as an include relationship.

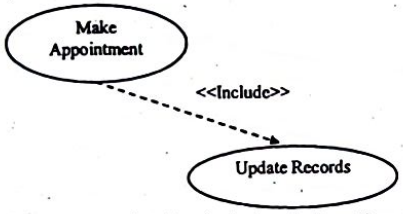


Fig: An example of an include relationship

Extends

In another form of interaction, a given use case (the extension) may extend another. This relationship indicates that the behavior of the extension use case may be inserted in the extended use case under some conditions.

An extend relationship is depicted with a directed arrow having a dotted line, similar to the include relationship. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow. The stereotype "<<extend>>" identifies the relationship as an extend relationship, as shown in the figure.

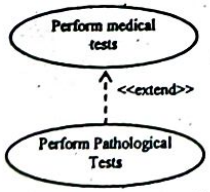
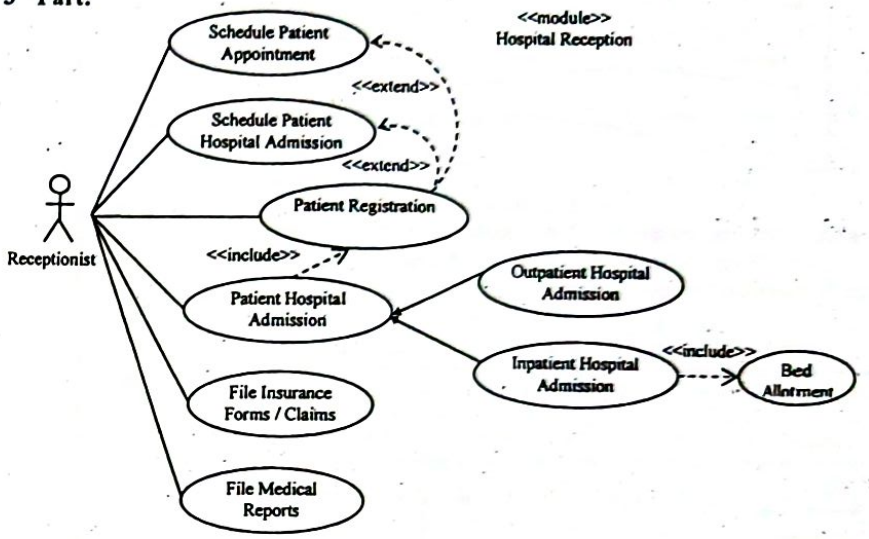


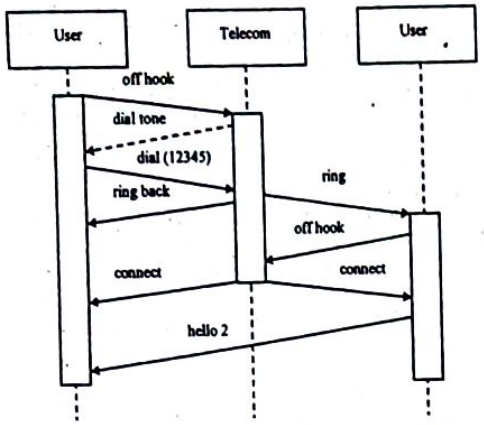
Figure shows an example of an extend relationship between the "Perform medical tests" (parent) and "Perform Pathological Tests" (child) use cases. The "Perform Pathological Tests" use case enhances the functionality of the "Perform medical tests" use case. Essentially, the "Perform Pathological Tests" use case is a specialized version of the generic "Perform medical tests" use case.

3rd Part:



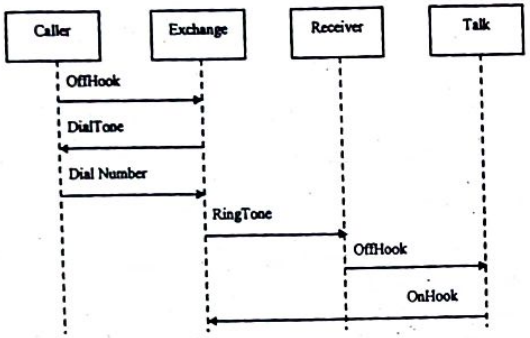
USE CASE Diagram with **"extends" and "includes"**

b) Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. A sequence diagram is an interaction diagram. From the name it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another. Interaction among the components of a system is very important from implementation and execution perspective. So Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.



Slanted messages take some time can model real-time systems. Sequence diagrams are an easy and intuitive way of describing the behavior of a system. A sequence diagram shows an interaction arranged in a time sequence.

Telephone Call



In UML, an activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process. Activity diagram describes the flow of control in a system. The flow can be sequential, concurrent or branched consisting of activities and links. Activities are nothing but the functions of a system. There may be multiple numbers of activities present to capture the entire flow in a system. Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed. Activity diagrams can be used for the following purposes:

- To describe a business process or a flow of work between users and the system.
- To describe the steps performed in a use case.
- To describe a method, function or operation in software.

- Drawing an activity diagram can help you improve a process.

Activity diagrams can be developed in various degrees of detail. They can be refined step by step. In the external view, activity diagrams, just like use case diagrams, exclusively represent business processes and activities from the outside perspective. Refining diagrams does not mean describing process details that are performed within the business system, which often leads to an unnoticed shift to the internal view:

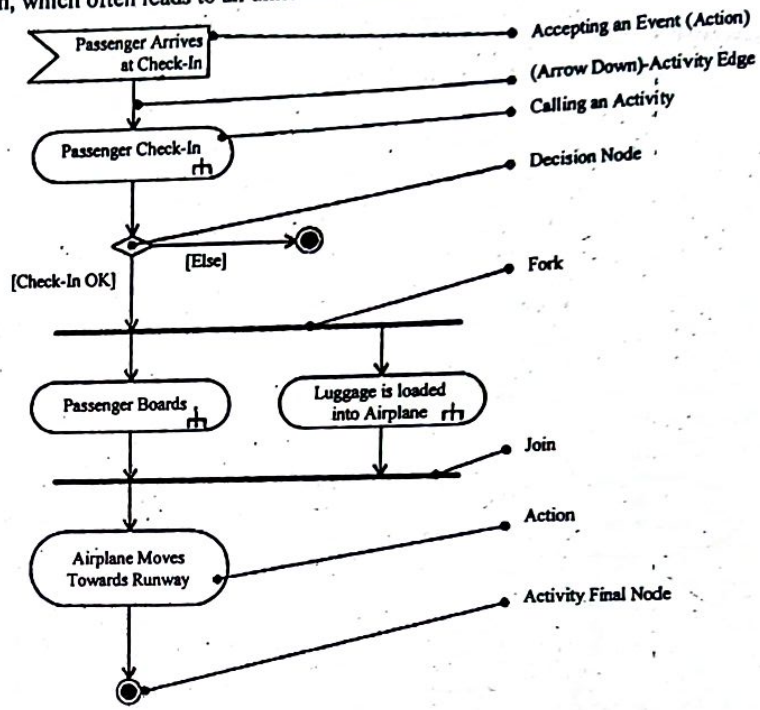


Fig: Activity diagram "Passenger Services" with a low level of detail ("High Level")

4. a) Explain when and why you will use PERT charts and when and why you will use Gantt charts while you are a project manager.
b) Consider a software project with 5 activities T1 to T5. Duration of 5 activities in weeks are 3, 2, 3, 5, 2 respectively. T2 and T4 can start when T1 is complete. T3 can start when T2 is complete. T5 can start when both T3 and T4 are complete. Draw activity network for the project. When is the latest start date of the activity T3? What is the float of the activity T4? Which activities are on the critical path? Draw the Gantt chart also. **[WBUT 2014]**

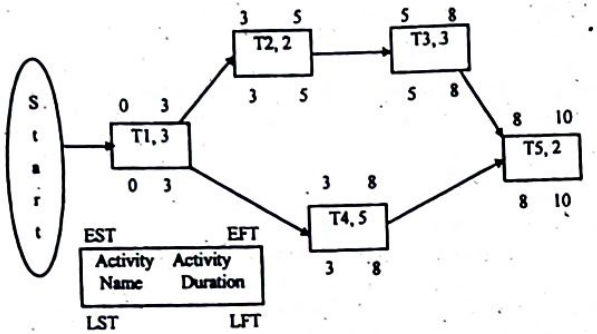
Answer:

a) One significant advantage of PERT charts is that all individual tasks and dependencies are shown

1) PERT stands for Project Evaluation and review Technique

2) PERT network produces probabilistic measures, where as Gantt only estimates.
3) A PERT chart displays the critical path for the overall project and the slack time
4) A Gantt chart offers a rapid overview
5) PERT and Gantt charts are not mutually exclusive techniques.

b)



LFT: Latest Finish Time; LST: Latest Start Time; EFT: Early Finish Time; EST: Early Start Time

Latest Start date of T3: 5 days

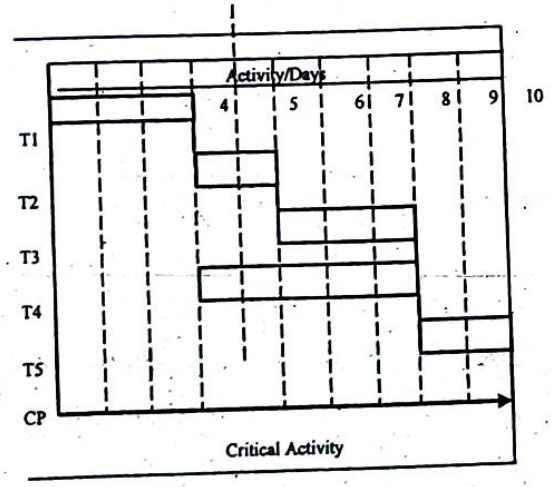
Float at T4= LST-EST=3-3=0 days

The two critical paths of the project are:

1. Start-T1-T2-T3-T5 = 10 Days
2. Start-T1-T4-T5=10 Days

As both the paths takes 10 days, both are critical, so slack at any activity is zero and all activities are considered to be critical.

Gantt chart:



5. a) Write down three advantages of decision trees over decision table.
b) Mention two situations where decision tables work best.
c) A bank has decided to adopt the following policy on deposits:
On deposit of Rs. 5,000/- and above and for three years or above the interest is 10%. On the same deposit for a period less than 3 years it is 8%. On deposits below Rs. 5,000 the interest is 6% regardless of the period of deposit.
Develop a decision tree and a decision table for the above process. Also express the above policy using structured English.
d) Distinguish between physical DFD and logical DFD with example of each.
[WBUT 2015]

Answer:

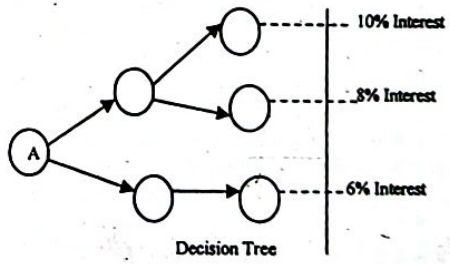
a) Some of the *advantages* of using decision trees are:

1. Decision rules are clearly structured
2. Decision making is consistent
3. Decision tree algorithms are intended for classification
4. Each branch of a decision tree corresponds to an outcome of the test and each external node denotes a class prediction.
5. At each node, the algorithm chooses the "best" attribute to partition the data into individual classes.
6. Decision tree is a method of documentation that is easily prepared, changed and Updated.
7. Communication between the System Analyst and the coder/programmer is easy.

In some cases decision tree is even advantageous over decision tables. Like, its construction is simple; diagrammatic presentations are clear and simple to take actions. Another disadvantage of decision table is that they do not scale up well as they are prone to redundancies.

b) The two occasions where DT's should be exploited are:
1. When there are "complex conditions, actions, and rules.
2. When there is many redundancies, and numerous contradictions within the process. Decision tables also have the ability to include every possible conditions.

c)



Decision Table:

	Condition Stub	1	2	3
a	Account Balance >=5000	Y	Y	
b	Account Balance < 5000			Y
c	Time >= 3 yrs	Y		
d	Time < 3 yrs		Y	
	Action Stub			
a	Interest @10%	X		
b	Interest @8%		X	
c	Interest @6%			X

d) Physical data flow diagrams show how the system operates or how the new system will be implemented.

- Physical data flow diagrams include
- Clarifying which processes are manual and which are automated
- Describing processes in greater detail
- Sequencing processes in the order they must be executed
- Temporary data stores and transaction files
- Specifying actual document and file names
- Controls to ensure accuracy and completeness
- Describes processes for adding, reading, changing, and deleting records
- CRUD is an acronym for Create, Read, Update, Delete
- A CRUD matrix shows which programs or processes add, read, update, or delete master file records.

Logical data flow diagrams show how the business operates. It shows the processes that would exist regardless of the type of system implemented

- The progression of creating data flow diagrams is
- Create a logical DFD of the current system
- Next add all the data and processes not in the current system which must be present in the new system
- Finally derive the physical data flow diagram for the new system
- Advantages of logical DFDs are
- Better communication with users
- More stable systems, since the design is based on a business framework
- Increased understanding of the business by analysts
- The system will have increased flexibility and be easier to maintain
- Elimination of redundancy

6. Draw a context diagram and top level DFD for the following system. Also provide relevant data dictionary.
The admission committee determines whether or not a student is admitted to graduate school. The associated dean determines the financial aid.

Students send applications to the graduate school. From other universities, the graduate school receives transcripts. Additionally, unrelated third parties provide letter of recommendation. The Graduate Management Admissions Council (GMAC) provides GMAT scores. Upon receipt of the above items, the graduate school prepares an application packet and enters the student's name in the pending application file. An acknowledgement letter is sent to the student. The graduate school sends the application packet to the admission committee. The admission committee reviews the student's credentials. In most cases the student's are accepted. The committee sends the applicant a letter explaining the result of review. A copy of the letter is also sent to the registrar who creates a student record in the registrar's system. The accepted student's file is sent to the associated dean for financial aid review. The associate dean keeps a list of available scholarships. Based on the review of the student's interests and the scholarship criteria a financial aid award letter is prepared, which again is sent to the student. Also, the controller's office receives a copy so that the proper bill can eventually be sent to the student once he/she registers for classes. The associate dean then updates the pending application file, closing out the student's record. Each month, the graduate school prepares a summary of how many applications have been received, approved and rejected. This report is sent to the university's president. **[WBUT 2015]**

Answer:

The following describes the application and approval process.

Students send applications to the graduate school. From other universities, the graduate school receives transcripts. Additionally, unrelated third parties provide letters of recommendation. The Graduate Management Admissions Council (GMAC) provides GMAT scores.

Upon receipt of the above items, the graduate school prepares an application packet and enters the student's name in the pending application file. An acknowledgment letter is sent to the student.

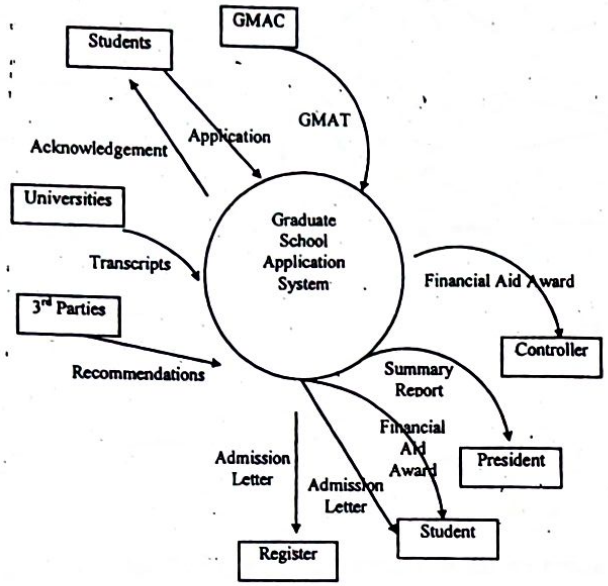
The Graduate school sends the application packet to the admissions committee. The admissions committee reviews the student's credentials. In most cases, students are accepted.

The committee sends the applicant a letter explaining the result of the review. A copy of the letter is also sent to the registrar who creates a student record in the registrar's system.

The accepted student's file s is sent to the associate dean for financial aid review. The associate dean keeps a list of available scholarships. Based on his review of the student's interests and the scholarship criteria a financial aid award letter is prepared, which again is sent to the student. Also, the controller's office receives a copy so the proper bill can eventually be sent to the student once he / she registers for classes.

The associate dean then updates the pending application file, closing out the student record.

each month, the graduate school prepares a summary of how many applications were received, approved, and rejected. They send this report to the university's president.



Context Diagram

7. Distinguish between object oriented design and function oriented design with proper examples. **[WBUT 2015]**

Answer:

In OOD state of information is not shared in a centralized data but is distributed among the objects of the system. In an employee pay-roll system, the following can be global data:

names of the employees,
employee code numbers,
basic salaries, etc.

Whereas, in object oriented systems data is distributed among different employee objects of the system. Objects communicate by message passing. One object may discover the state of information of another object by interrogating it. Of course, somewhere or other the functions must be implemented the functions are usually associated with specific real-world entities (objects) directly access only part of the system state information.

Function-oriented techniques group functions together if as a group, they constitute a higher level function. On the other hand, object-oriented techniques group functions together on the basis of the data they operate on. **In the function-oriented program the**

system state is centralized several functions accessing these data are defined. In the object oriented program, the state information is distributed among various objects. Use OOD to design the classes then applies top-down function oriented techniques to design the internal methods of classes. Though outwardly a system may appear to have been developed in an object oriented fashion, but inside each class there s a small hierarchy of functions designed in a top-down manner.

8. a) What is the difference between a flow chart and a structure chart?
b) List the points of a simplified design process. **[WBUT 2016]**

Answer:

1. A structure chart is used to show how modules of a computer program relate to each other. A flow chart shows a single module in detail
2. A structure chart can represent an entire program with modules whereas a flowchart can represent a module in a program.
3. In structure chart, rectangles are used to represent modules whereas a flow chart is a pictorial representation of an algorithm that uses boxes of different shapes to denote terminal, process, input/output, flow line, connector, decision etc outlines are usually used.

b) The steps of the simplified engineering design process are to:

1. Define the Problem
2. Do Background Research
3. Specify Requirements
4. Brainstorm Solutions
5. Choose the Best Solution
6. Do Development Work
7. Build a Prototype
8. Test and Redesign

Engineers/Developers do not always follow the engineering design process steps in order, one after another. It is very common to design something, test it, find a problem, and then go back to an earlier step to make a modification or change the design. This way of working is called **iteration**, and it is likely that the process will produce the identical output.

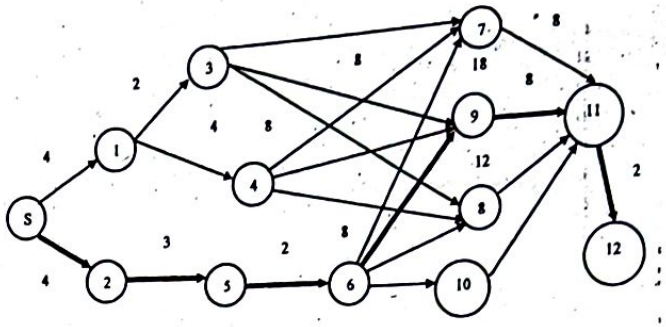
9. Suppose you are the Project Manager of a Software Project that consists of following Activities in the table and you have to draw the activity network and find the critical tasks of the project.
Draw the Gantt chart of the Project. (Consider Resources allocation will start from 12th March, 2010). **[WBUT 2017]**

Activity No.	Activity Name	Duration (weeks)	Immediate Predecessor
1.	Obtain Requirements	4	–
2.	Analyze Operations	4	–
3.	Define Subsystems	2	1
4.	Develop Database	4	1

5.	Make Decision Analysis	3	2
6.	Identify Constraints	2	5
7.	Build Module 1	8	3, 4, 6
8.	Build Module 2	12	3, 4, 6
9.	Build Module 3	18	3, 4, 6
10.	Write Report	10	6
11.	Integration and Testing	8	7, 8, 9
12.	Implementation	2	10, 11

Answer:

Task Name	Durat	Start	Finish	Predecessors
obtain requirement	4 wks	Fri 12-03-10	Thu 08-04-10	
Analyze operations	4 wks	Fri 12-03-10	Thu 08-04-10	
Define Subsystems	2 wks	Fri 09-04-10	Thu 22-04-10	1
Develop Database	4 wks	Fri 09-04-10	Thu 06-05-10	1
Make decision Analysis	3 wks	Fri 09-04-10	Thu 29-04-10	2
Identify Constraints	2 wks	Fri 30-04-10	Thu 13-05-10	5
Build Module 1	8 wks	Fri 14-05-10	Thu 08-07-10	3,4,6
Build Module 2	12 wks	Fri 14-05-10	Thu 05-08-10	3,4,6
Build Module 3	18 wks	Fri 14-05-10	Thu 16-09-10	3,4,6
write Report	10 wks	Fri 14-05-10	Thu 22-07-10	6
Integration and Testing	8 wks	Fri 17-09-10	Thu 11-11-10	7,8,9
Implementation	2 wks	Fri 12-11-10	Thu 25-11-10	10,11

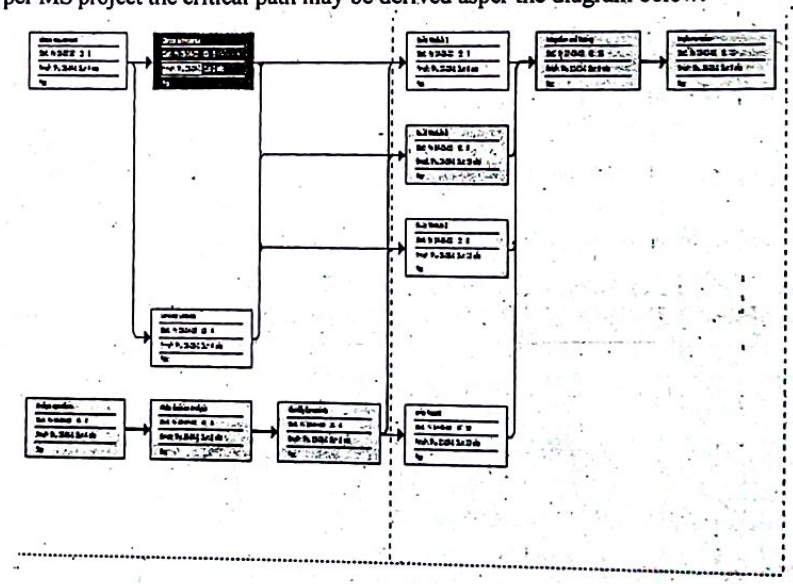


List of paths:

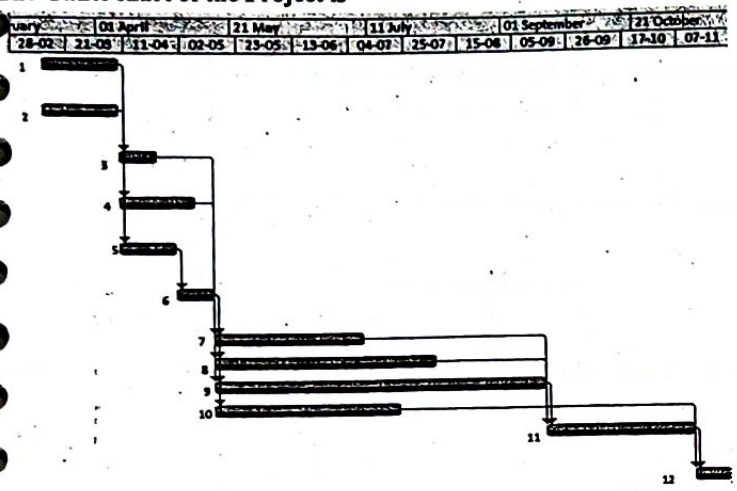
1. S-1-3-7-11-12 = 4+2+8+8+2= 24 Wks
2. S-1-3-9-11-12 = 4+2+12+8+2=28 Wks
3. S-1-3-8-11-12 =4+2+18+8+2=34 Wks
4. S-1-4-7-11-12 =4+4+8+8+2=26 Wks
5. S-1-4-9-11-12 = 4+4+18+8+2=36 Wks
6. S-1-4-8-11-12 =4+4+12+8+2=30 Wks
7. S-2-5-6-7-11-12 =4+3+2+8+8+2=27 Wks
8. S-2-5-6-9-11-12 =4+3+2+18+8+2=37 Wks
9. S-2-5-6-8-11-12 =4+3+2+12+8+2=31 Wks

The path '8' is the longest path of the project i.e. the critical path of the project. Total duration along the critical path is 37 weeks. It starts at 12/03/10 and will finish at 25/11/10.

As per MS project the critical path may be derived asper the diagram below:



The Gantt chart of the Project is



10. Consider the following C program: **[WBUT 2017]**

```
Int Compute_gcd (x, y)
int x, y:
{
        while (xl = y)
                if (x > y) then x = x – y;
                else y = y – y;
        return x;
}
```

a) Find out the estimated length, Program vocabulary, Program volume, Effort, Time. Comment on the technique that you use to solve the problem.
b) Compare Halstead's length and volume measures of size with the LOC measure.

Answer:

a) List of operators and operands are:

Operators	Occurrences	Operands	Occurrences
Int	2		
Compute_gcd	1	x	7
()	3	y	8
,	2		
{}	1		
while	1		
if	1		
Then	1		
1 =	1		
>	1		
=	2		

-	2		
else	1		
return	1		
;	4		
η1=15	N1=24	η2=2	N2=15

Program length: N=N1+N2=39
Program vocabulary: $\eta = \eta 1 + \eta 2 = 17$
Estimated length: $=15 \log_2 15 + 2 \log_2 2 = 62$
Prog.Volume $V = N * \log_2 \eta = 39 \times \log_2 (17) = 39 \times 4.087 = 159.411$ bits
Estimated program level=$(2/\eta 1) \times (\eta 2/N2) = (2/15) \times (2/15) = 0.1333$
Effort:=V/ Estimated program level= 159.411/0.1333=1195.88
Time= E/β=(1195.88/18) [*β=18 give best result in Halastead's experiments]
=66.44 Seconds

Halstead complexity measures are software metrics introduced by Halstead in 1977 as part of his discourse on establishing an empirical science of software development. The Halstead measures are based on four scalar numbers derived directly from a program's source code. These metrics are computed statistically from the code. The goal is to identify measurable properties of software, and the relations between them.

b) Halstead Length (N): The length of your program, computed N1+N2.
As per our example above Halstead Length is 39

Halstead Volume
N is program length
N = total number of operators N1 + total number of operands N2
n is program vocabulary
N = number of distinct operators n1 + number of distinct operands n2

Lines of Code (LOC)
LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored. Determining the LOC count at the end of a project is a very simple job. However, accurate estimation of the LOC count at the beginning of a project is very difficult. In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted. To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.

Lines-of-code (LOC) metrics offer a gross measure of code, but do not measure content well. However, LOC in combination with Halstead measures may help relate program size to functionality

11. What is Cohesion? Explain the cohesion classification with respect to software design. **[WBUT 2018]**

Answer: ***Refer to Question No. 13(b) [1st Part] of Long Answer Type Questions.***

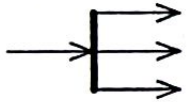
12. a) What do you mean by forking and joining in Activity Diagram? What is a swimlane?
b) What are extend and include in use case diagram?
c) What are dependency, aggregation and composition in use case diagram? Explain with example. **[WBUT 2018]**

Answer:

a) 1st Part:

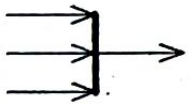
Fork node is a control node that has one incoming edge and multiple outgoing edges and is used to split incoming flow into multiple concurrent flows. Fork nodes are introduced to support parallelism in activities. As compared to UML 1.5, UML 2.0 activity forks model unrestricted parallelism.

The notation for a fork node is a line segment with a single activity edge entering it, and two or more edges leaving it.



Join node is a control node that has multiple incoming edges and one outgoing edge and is used to synchronize incoming concurrent flows. Join nodes are introduced to support parallelism in activities.

The notation for a join node is a line segment with several activity edges entering it, and only one edge leaving it.



2nd Part: A swimlane (or swimlane diagram) is used in process flow diagrams, or flowcharts, that visually distinguishes job sharing and responsibilities for sub-processes of a business process. Swimlanes may be arranged either horizontally or vertically.

b) ***Refer to Question No. 3(a) (2nd Part) of Long Answer Type Questions.***

c) Dependency is defined as a relation between two classes, where one class depends on another class but another class may or not may depend on the first class. So any change in

one of the classes may affect the functionality of the other class that depends on the first one.

Aggregation is the same as association but with an additional point that there is an ownership of the instances, unlike association where there was no ownership of the instances. Aggregation is also referred to as a Weak Association and is represented by the following symbol in UML representation. let's add another class named Department to our example explained above.

The relation between Teacher and Department then conceptually, a Department can have multiple Teachers associated with it but each Teacher can belong to only one Department at a time.

Composition is a special case of aggregation. In more specific manner, a restricted aggregation is called Composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called Composition.

Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

13. Write short notes on the following:
a) Characteristics of a good user interface **[WBUT 2013]**
b) Cohesion and Coupling **[WBUT 2013]**
c) Decision Tree and Decision Table **[WBUT 2014]**
d) Work Break down Structure & Utility of PERT Chart **[WBUT 2017]**

Answer:

a) Characteristics of a good user interface:

A graphical user interface or GUI, sometimes pronounced "gooey" is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.

GUIs display visual elements such as icons, windows and other gadgets. The precursor to GUIs was invented by researchers at the Stanford Research Institute i[th] the development and use of text-based hyperlinks manipulated with a mouse for the On-Line System. The concept of hyperlinks was further refined and extended to graphics by researchers at Xerox PARC.

Examples of systems that support GUIs are:

- Mac OS,
- Microsoft Windows,
- NEXTSTEP
- X Window System.

Individual Elements of User Interfaces are:

- About box
- Dialog box
- Icon
- Balloon help

- Button
- Check box
- Combo box

b) Cohesion and Coupling:

Cohesion: Most researchers and engineers agree that a good software design implies clean decomposition of the problem into modules, and the neat arrangement of these modules in a hierarchy. The primary characteristics of neat module decomposition are high cohesion and low coupling. Cohesion is a measure of functional strength of a module. A module having high cohesion and low coupling is, said to be functionally independent of other modules. By the term functional independence, we mean that a cohesive module performs a single task or function. A functionally independent module has minimal interaction with other modules.

Classification of Cohesion: The different classes of cohesion that a module may possess are:

- Coincidental cohesion
- Logical cohesion
- Communications cohesion
- Functional cohesion

Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules. A module having high cohesion and low coupling is said to be functionally independent of other modules. If two modules interchange large amounts of data, then they are highly interdependent. The degree of coupling between two modules depends on their interface complexity. The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

Classification of Coupling: Even if there are no techniques to precisely and quantitatively estimate the coupling between two modules, classification of the different types of coupling will help to quantitatively estimate the degree of coupling between two modules, rive types of coupling can occur between any two modules.

- Data coupling
- Stamp coupling
- Control coupling'
- Common coupling
- Content coupling

c) Decision Tree and Decision Table:

A decision table is a tabular representation of a logical procedure by means of a set of conditions and related actions in the form of a matrix of rows and columns. Decision table is useful for specifying complex policies and decision-making rules. Decision table is made up of four sections presented below:

a. Condition Stub
b. Action Stub
c. Condition Entries
d. Action Entries

A Decision Table (DT) is a visual means for showing how a rule/or set of rules applies to repetitive situations. It is a tool of the programmer as well as that of the systems analyst. It can be used to supplement the flowchart. It is a 2×2 matrix. Where quadrant (1,1) represents condition stub. Quadrant (2,1) represents actions stub. Quadrant (1,2) indicates condition entry and (2,2) represents action entry. The physical layout of a decision table is as given below:

Header (H)	Rule (R) Identifiers
Condition stub/Statements	Condition Entry
Action Stub/statements	Action Entry
Notes: (NB)	

Decision table is suitable for complex and large size data analysis.

A decision tree is a diagrammatic representation of conditions and actions sequentially on priority basis that resembles branches on a tree. The root of the tree is set on the left of the diagram and it is the starting point of the decision sequence. The particular branch to be followed depends on the conditions that exist and the decision to be made. The following is a pictorial representation of a decision tree:

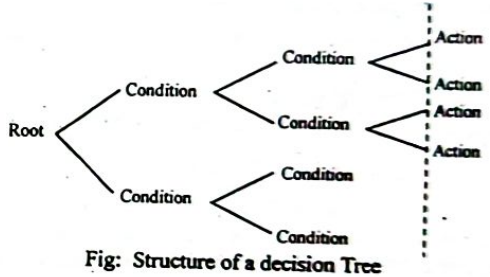


Fig: Structure of a decision Tree

Example:

A bank will grant loans under the following conditions:

1. Loan will be granted to a customer having a bank account and has no outstanding loan.
2. If a customer has an account with the bank but some amount is outstanding from previous loans, then loan will be granted if special management approval is obtained.
3. Reject loan applications in all other cases.

The decision tree will look like:

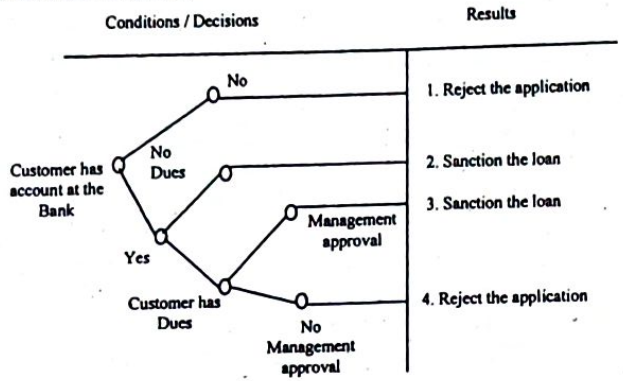
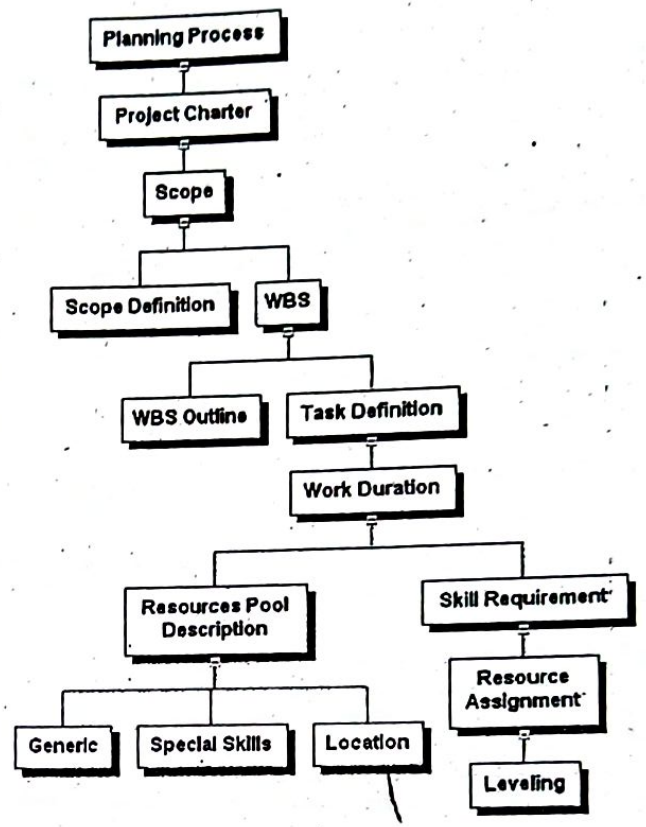


Fig: Decision Tree

d) **WBS:**

1. WBS is a deliverable-oriented hierarchical decomposition of the work to be executed by the project team, to accomplish the project objectives and create the required deliverables
2. WBS represents the work specified in the current approved project scope/charter statement
3. Components comprising the WBS assist the stakeholders in viewing the deliverables of the project



PERT:

1. PERT stands for Program Evaluation and Review Technique
2. A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project
3. PERTs become very complex very quickly, so the key to usage is to plan and manage small chunks of work

In conclusion it is presented as:

WBS Charts aids the team to visualize the plan

- Reduces inconsistencies in the project plan
- Reduces duplicated tasks
- Intuitive
- PERT Charts reinforce sequencing (bi-directional)
- Locates orphan tasks

- Promotes fixing obvious missing predecessors and successors
- Identifies wrong owners & process groups
- Opportunities to shorten timelines

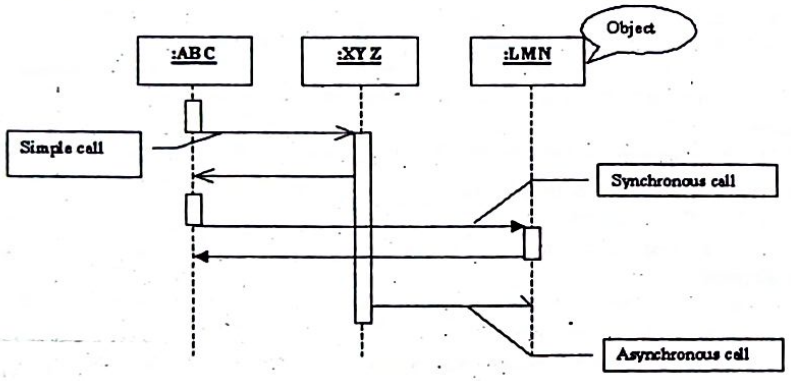
14. What is UML? What is the purpose of UML? Briefly explain the 'Sequence Diagram'. **[MODEL QUESTION]**

Answer:

UML (Unified Modeling Language) is an object-oriented analysis and design language defined by the Object Management Group (OMG). UML is used in the software engineering field and describes a programming language used as a graphical designation to create an abstract model that can be used in a system (UML model).

UML is primarily used to construct software intensive systems. In today's information technology industry, the need for efficiency and rapid software development in short periods of time has become a programmer's main challenge. Building complex software applications is a difficult process and learning and utilizing UML assists the developer in seeing these processes through.

A sequence diagram is made up of objects and messages. Objects are represented exactly how they have been represented in all UML diagrams—as rectangles with the underlined class name within the rectangle. A skeleton sequence diagram is shown in figure below. We shall discuss each of these elements in the next section:



15. a) What are the different elements of object model? Describe each of them briefly.

b) What is UML? **[MODEL QUESTION]**

Answer:

a) There are five main kinds of programming styles, listed here with the kinds of abstractions they employ:

1. Procedure-oriented	Algorithms
2. Object-oriented	Classes and objects

3. Logic-oriented	Goals, often expressed in a predicate calculus
4. Rule-oriented	If-then rules
5. Constraint-oriented	Invariant relationship

There is no single programming style that is best for all kinds of applications. For example, rule-oriented programming would be best suited for the design of a knowledge base, and procedure-oriented programming would be best for the design of computation-intense operations. From our experience the object-oriented style is best suited tc the broadest set of applications; indeed, this programming paradigm often serves as the architectural framework in which we employ other paradigms.

Each of these styles of programming is based on its own conceptual framework. Each requires a different mindset, a different way of thinking about the problem. For all things object-oriented, the conceptual framework is the object model. There are four major elements of this model:

1. Abstraction
2. Encapsulation
3. Modularity
4. Hierarchy

By major, we mean that a model without any one of these elements is not object-oriented. There are three major elements of Object Model:

1. Typing
2. Concurrency
3. Persistence

By minor, we mean that each of these elements is a useful, but not essential, part of the object model.

Without this conceptual framework, one may be programming in a language such as Smalltalk, Object Pascal, C++, Eiffel, or Ada, but your design is going to smell like a FORTRAN, Pascal, of C application.

b) The Unified Modeling Language (UML) is a standard language used for modeling the various components, their behavior, their relationship and the way they interact within the software system or any other non- software systems. These components are used for visualizing, analyzing and documenting the artifacts of the software system. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. That UML plays a significant role in software development process, because it uses graphical notations to express the modeling of software projects.

16. Define the following terms: **[MODEL QUESTION]**

i) Sequence diagram

ii) Links

iii) State chart diagram

iv) Collaboration diagram

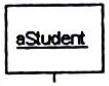
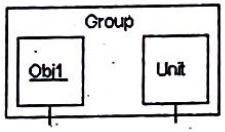
Answer:

(i) Sequence Diagram:

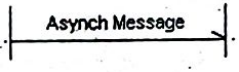
UML sequence diagrams are used to represent or model the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

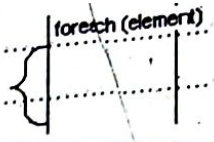
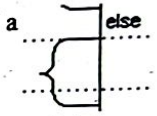
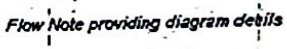
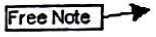
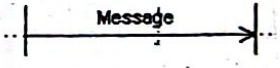
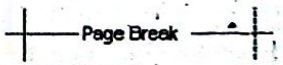
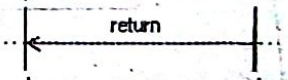
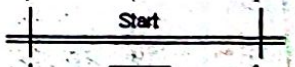
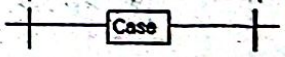
Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.

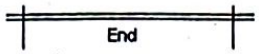
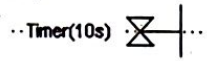
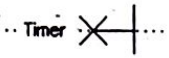
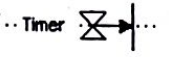
Although UML sequence diagrams are typically used to describe object-oriented software systems, they are also extremely useful as system engineering tools to design system architectures, in business process engineering as process flow diagrams, as message sequence charts and call flows for telecom/wireless system design, and for protocol stack design and analysis.

Actor	Represents an external person or entity that interacts with the system	
Object	Represents an object in the system or one of its components	
Unit	Represents a subsystem, component, unit, or other logical entity in the system (may or may not be implemented by objects)	
Separator	Represents an interface or boundary between subsystems, components For units (e.g., air interface, Internet, network)	
Group	Groups related header elements into subsystems or components	

Sequence Diagram Body Elements

Action	Represents an action taken by an actor, object or unit	
Asynchronous Message	An asynchronous message between header elements	

Block	A block representing a loop or conditional for a particular header element	
Call Message	A call (procedure) message between header elements	
CreateMessage	A "create" message that creates a header element (represented by lifeline going from dashed to solid pattern)	
Destroy Element	Represents the destruction of a header element	
Destroy Message	Represents the destruction of a header element as a result of a call from another element	
Diagram Link	Represents a portion of a diagram being treated as a functional block. Similar to a procedure or function call that abstracts functionality or details not shown at this level. Can optionally be linked to another diagram for elaboration.	
Else Block	Represents an "else" block portion of a diagram block	
Flow Note	Documentation note that is automatically formatted to flow after previous elements	
Free Note	Documentation note that is free-flowing and can be placed anywhere in the diagram (can also be anchored relative to a flow element)	
Message	A simple message between header elements.	
Page Break	A page break in the diagram	
Return Message	A return message between header elements	
Scenario Start	Start of a scenario (set of alternatives)	
Scenario Case	Start of an alternative or case in a scenario	

Scenario End	End of a scenario	
State	A state change for a header element	
Steady State	A steady state in the system	
Timer Start	Start of a timer for a particular header element	
Timer Stop	Stop of a timer for a particular header element	
Timer Expiration	Expiration of a timer for a particular header element	

(ii) Links:

A UML link is run-time relationship between instances of classifiers. A typical uni-directional link requires the one instance to know about, and thus depend, upon the other, but this is not required. Likewise, a bi-directional link requires that both instances may traverse to each other, but this also does not require dependency.

(iii) State Chart Diagram:

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a conponent/object of a system.

A state chart diagram describes a state machine. Now to clarify it state machine can be defned as a machine which defines different states of an object and these states are controlled by external or internal events.

Actvity diagram is a special kind of a State chart diagram. As State chart diagram defnes states it is used to model lifetime of an object.

State chart diagram is one of the five UML diagrams used to model dynamic nature of a system. They define different states of an object during its lifetime. And these states are chnged by events. So State chart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events. State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of State chart diagram is to model life time of anobject from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model reactive system.

Following are the main purposes of using State chart diagrams:

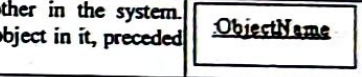
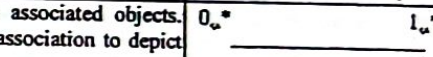
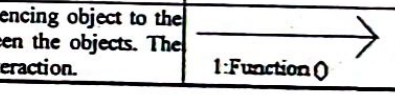
- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model states of an object.

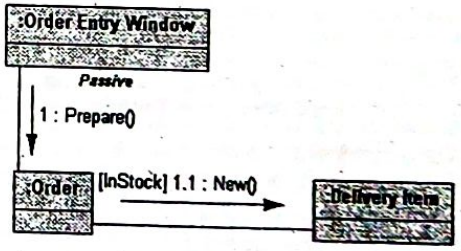
(iv) Collaboration Diagram:

A Collaboration diagram is very similar to a Sequence diagram in the purpose it achieves; in other words, it shows the dynamic interaction of the objects in a system. A distinguishing feature of a Collaboration diagram is that it shows the objects and their association with other objects in the system apart from how they interact with each other. The association between objects is not represented in a Sequence diagram.

A Collaboration diagram is easily represented by modeling objects in a system and representing the associations between the objects as links. The interaction between the objects is denoted by arrows. To identify the sequence of invocation of these objects, a number is placed next to each of these arrows.

A Collaboration diagram consists of the following elements:

Element and its description	Symbol
Object: The objects interacting with each other in the system. Depicted by a rectangle with the name of the object in it, preceded by a colon and underlined.	
Relation/Association: A link connecting the associated objects. Qualifiers can be placed on either end of the association to depict cardinality.	
Messages: An arrow pointing from the commencing object to the destination object shows the interaction between the objects. The number represents the order/sequence of this interaction.	



CODING, TESTING & QUALITY ASSURANCE

Multiple Choice Type Questions

1. Alpha testing is done by [WBUT 2013]
a) customer b) tester c) developer d) none of these
Answer: (a)

2. Beta testing is done by [WBUT 2013]
a) the developers pages b) blocks
c) holes d) frames
Answer: (a)

3. Testing the software is basically [WBUT 2013]
a) verification b) validation
c) both (a) and (b) d) none of these
Answer: (c)

4. Functional testing is known as [WBUT 2013]
a) structural testing b) behavior testing
c) registration testing d) none of these
Answer: (b)

5. System test is performed by [WBUT 2013]
a) end users b) programmers c) system analyst d) none of these
Answer: (d)

6. Equivalence class partitioning is followed in [WBUT 2013]
a) white box testing b) black box testing
c) both (a) and (b) d) none of these
Answer: (c)

7. In Integration Testing approach, where all modules making up a system are integrated in a single step is known as – [WBUT 2014]
a) top-down Integration Testing b) bottom-up Integration Testing
c) big-band Integration Testing d) mixed Integration Testing
Answer: (b)

8. Alpha-testing is done by [WBUT 2014]
a) the development team b) a friendly set of customers
c) the customer himself d) none of these
Answer: (a)

9. One way to improve readability in coding is to [WBUT 2015]
a) avoid goto statements
b) name variables and functions according to their use
c) modularize the program
d) none of these
Answer: (a)

10. The type of failure that occurs for all input values while invoking a function of the system is [WBUT 2015]
a) transient failure b) permanent failure
c) recoverable failure d) unrecoverable failure
Answer: (b)

11. Function point describes [WBUT 2015, 2018]
a) the SRS document b) the test plans
c) the functional decomposition
d) the size of a software product directly from its specification
Answer: (d)

12. Efforts are measured in terms of [WBUT 2015]
a) Person-months b) Persons c) Rupees d) Months
Answer: (a)

13. The cyclomatic complexity of the following program fragment is: [WBUT 2015]
```
int gcd (int x, int y)
while (xl = y)
if (x > y) then
x = x – y;
else y = y – x;
}
return x;
}
```
a) 2 b) 3 c) 4 d) 5
Answer: (b)

14. Coding and testing are done in which of the following manner? [WBUT 2017]
a) Adhoc b) Top-clown
c) Cross sectional d) Bottom-up
Answer: (a)

15. The largest percentage of total life cycle cost of software is [WBUT 2017]
a) Design cost b) Coding cost
c) Maintenance cost d) Testing cost
Answer: (a)

ain difference between program testing and system testing is [WBUT 2017]

a) System testing focuses on testing the interfaces between programs, program testing focuses on individual programs
b) Program testing is more comprehensive than system testing
c) System testing is tough and program testing is easy
d) none of these

wer: (a)

7. Decision support systems are used to [WBUT 2017]

a) Evaluate and analyze the mission of organization
b) To perform accounting
c) Perform multiple activities simultaneously
d) none of these

wer: (a)

8. Tracking the correspondence between the design component and the SRS is nown as [WBUT 2018]

a) availability
b) traceability
c) maintainability
d) reliability

nswer: (b)

9. System Testing is performed by a friendly set of customer is known as [WBUT 2018]

a) alpha testing
b) beta testing
c) performance testing
d) usability testing

nswer: (b)

0. The potential risks are best detected by [WBUT 2018]

a) spiral model
b) waterfall model
c) incremental model
d) prototyping model

nswer: (a)

1. The most desirable form of cohesion is [WBUT 2018]

a) sequential cohesion
b) procedural cohesion
c) coincidental cohesion
d) functional cohesion

nswer: (d)

2. The best type of coupling is [WBUT 2018]

a) coincidental
b) logical
c) informational
d) functional

wer: (d)

3. Big Bang Integration testing is useful for project with [WBUT 2018]

a) smaller number of modules
b) large number of modules
c) average number of modules
d) none of these

nswer: (d)

Short Answer Type Questions

1. What is Mutation Testing? [WBUT 2014]
Distinguish between White-box Testing and Black-box testing? [WBUT 2014]
OR,
What is the difference between black-box and white-box testings? [WBUT 2016]

Answer:

1st Part:

A *mutant* (or mutation) is a computer program that has been purposely altered from its original version Mutation Testing is done on the application where bugs are purposely added to it. ***Mutation Testing*** is NOT a testing strategy like *path* or *data-flow* testing. It does not outline test data selection criteria. It should be used in conjunction with traditional testing techniques, not instead of them. For example, one mutation operator changes one binary operator (e.g. &&) to another (e.g. ||) in an attempt to create a fault variant of the program.

A test case is ***adequate*** if it is useful in detecting faults in a program. The major drawback of mutation testing is the cost of generating the mutants and executing each test case against them.

2nd Part:

While black box and white box testing are called by different names, they basically involve looking at the structure of the code (white box) or looking at the functionality of the code (black box). There is some question as to which approach is the most effective or efficient. There is also an argument as to how much should be involved in white box testing?

White box testing tends to involve coverage, while black box testing tends to involve functionality. An inherent weakness of white box testing is that we tend to find errors when the function is partly wrong, but if it is a wrong function programmed correctly, it is almost impossible to detect with white box testing.

2. Distinguish between software verification and software validation. When during the life cycle verification and validation performed? [WBUT 2014]

Answer:

1st Part:

Like testing, verification is also intended to find errors. Executing a program in a simulated environment performs it. Validation refers to the process of using software in a live environment to find errors. When commercial systems are developed with the main aim of distributing them to dealers for sale purposes, they first go through verification, sometimes called alpha testing. The feedback from the validation phase generally brings some changes in the software to deal with errors and failures that are uncovered. Then a set of user sites is selected for putting the system into use on a live basis. These beta test sites use the system in day-to-day activities; they process live transactions and produce normal system output. Validation may continue for several months. During the course of

validating the system, failure may occur and the software will be changed. Continued use may bring more failures and the need for still more changes.

2nd Part:

Like testing, verification is also intended to find errors. Executing a program in a simulated environment performs it. Validation refers to the process of using software in a live environment to find errors. When commercial systems are developed with the main aim of distributing them to dealers for sale purposes, they first go through verification, sometimes called alpha testing. The feedback from the validation phase generally brings some changes in the software to deal with errors and failures that are uncovered. Then a set of user sites is selected for putting the system into use on a live basis. These beta test sites use the system in day-to-day activities; they process live transactions and produce normal system output. Validation may continue for several months. During the course of validating the system, failure may occur and the software will be changed. Continued use may bring more failures and the need for still more changes.

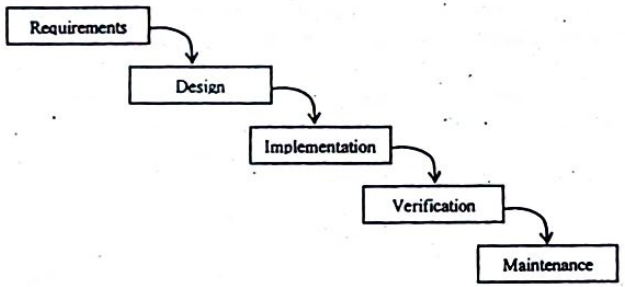
3. What is "Top-Down and Bottom-Up Design" approach? [WBUT 2014]

OR,

Explain top-down and bottom-up design. [WBUT 2015]

Answer:

Top-Down Design is a process, where refine the design at each step and then decompose, divide and conquer. In a bottom-up design approach it is just opposite. It starts with parts and then composed.



In case of simple programs, a simple waterfall likes process,

- But to use this process, one needs to be sure that the requirements are fixed and well understood!
  - Many software problems are not like that
  - Often customer refines the requirements when try to deliver the initial solution
- With Top-Down approach it is harder to test early because parts needed may not have been designed yet
- With Bottom-Up, you may end up needing things different from how you built them

4. What are different levels of Testing and their goals? [WBUT 2015]

Answer:

Testing is used to detect faults introduced during specification and design stages as well as coding stages. Due to this, different levels of testing are used in the testing process.

Each level of testing aims to test different aspects of the system. The basic level of testing is:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

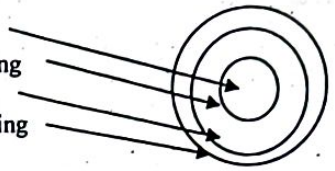


Fig: Levels of Testing

Testing starts at unit testing, then all the units are integrated and testing like Big Bang tests are performed. Next System Tests are performed and finally acceptance tests are performed before implementation of the system.

5. Distinguish between error and failure. Which of the two is detected by testing? Justify your answer. [WBUT 2017]

Answer:

According to Myers (1979) **testing** is the process of analyzing a program with the intent of finding **errors**. The term error is used to refer to the discrepancy between computed, observed, or measured value and the true or specified value. In other words, it is the difference between the actual output of software and correct output. Fault is a condition that causes a system to fail in performing its required function.

Failure: Failure is the inability of a system to perform a required function according to its specifications. A software failure occurs if the behavior of the software is different from the specified behavior.

In software development, errors can be cropped up at any stage during development. However, no method is perfect, and it is expected that some errors of the earlier phases may finally reaches to the coding phase. This is because most of the verification methods of earlier phases are manual. Hence the code developed during coding activity is likely to have some **requirement errors and design errors, in addition to errors introduced during the coding activity.**

During testing, the program to be tested is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as expected. From this it is clear that **testing is used to find out errors.** Also, the success of the testing process clearly depends upon the test cases used. Testing is a complex process. In order to make the process simpler, the testing activities are broken into smaller activities. Due to this, for a project, **incremental testing** is generally performed. In incremental testing process, the system is broken into set of subsystems and these

ubsystems are tested separately before integrating them to form the system for system g.

hase Containment of Errors: The phase containment of errors is a process of finding . Fault is identified during the review phase. Defect is a fault introduced from the rior phases but escaped from the previous inspections and/or reviews. During the project xecution; fault may be found at a later stage of the project caused relatively higher cost f correction. It is a best practice to promote an environment where the project evelopment team attempts to identify fault as early as possible and fix it promptly. actors of the total number of errors and the total number of faults found from a work roduct is closer to the value of 100%, the more effective of the inspection and/or review n a particular work product.

. Spiral model is not a panacea'; explain the meaning of the statement. Also state hy spiral is called a Meta-model. [WBUT 2018]

nswer:

he English dictionary says the word panacea as the solution of everything. The spiral el is a risk-driven software development process model. Based on the unique risk atterns of a given project, the spiral model guides a team to adopt elements of one or process models, such as incremental, waterfall, or evolutionary prototyping. When, e say it is not a panacea means it can't produce solutions for any type of software evelopment.

piral model is called Meta model because it comprehends all other life cycle models. It alled so also because of the way it comprises of other models of SDLC. E.g. both aterfall and prototype models are used in it.

. State characteristics of feature point metrics. [WBUT 2018]

nswer: *Refer to Question No. 2(3rd Part) of Long Answer type Questions.*

. What do you mean by an object lifeline and focus of control and which diagram required and why? [WBUT 2018]

nswer:

ML diagram, such as sequence diagram requires the object lifeline and focus of ontrol.

UML diagrams, such as sequence or communication diagrams, object lifelines epresent the objects that participate in an interaction. For example, in a banking scenario, elines can represent objects such as a bank system or customer. Each instance in an teraction is represented by a lifeline.

ocus of control (FOC) is used in sequence diagrams to show the period of time during hich an object performs an action. FOC is rendered as a thin, rectangular object that sits n top of object lifelines. The top of the FOC rectangle coincides with the receipt of a essage.

What are the differences between Alpha Testing and Beta Testing?
[MODEL QUESTION]

Answer:

Alpha testing: Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site, but outside the development organization.

Beta testing: Operational testing by potential and/or existing users/customers at an external site not otherwise involved with the developers, to determine whether or not a component or system satisfies the user/customer needs & fits within the business processes. Beta testing is often employed as a form of external acceptance testing for off the shelf software in order to acquire feedback from the market.

10. Why do we try to minimize coupling and maximize cohesion?
[MODEL QUESTION]

Answer:

Coupling and cohesion are often used as opposite ends of a scale in measuring how "good" a piece of software is. They are very common metrics for measuring the quality of object-oriented code.

Cohesion describes how "focused" a piece of software is. A highly-cohesive system is one in which all procedures in a given module work together towards some end goal. High cohesion is often characterized by high readability and maintainability.

Coupling describes how reliant a given piece of software is on other modules. A highly coupled system in one in which the procedures in one module can directly access elements of another module. A low coupled system in one in which the procedures of one module can only interact with the procedures of another through an interface channel. Highly coupled systems are often characterized by code that is difficult to read and maintain (the reason cohesion and coupling are often used as opposites).

11. What is acceptance testing? Compare Top-down and Bottom-up integration testings.
[MODEL QUESTION]

Answer:

Acceptance testing is black-box testing performed on a system (for example: a piece of software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery. It is also known as functional testing, black-box testing, QA testing, application testing, confidence testing, final testing, validation testing, or factory acceptance testing.

Software developers often distinguish acceptance testing by the system provider from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership. In the case of software, acceptance testing performed by the customer is known as user acceptance testing (UAT), end-user testing, site (acceptance) testing, or field (acceptance) testing.

Top down Approach: It is nothing but, checking the communication between the Main Module to the Sub module.

Example: If A, B & C are three Module, Assume A is the main module and B, C are the sub modules. Checking the communication from main module (A) to sub modules B & C are nothing but Top down approach.

tub: In above example, assume that, if Sub module B is under construction. At that me, the Developer take the help of a Temporary program (called as Stub) to check the ommunication between the modules.

ottom Up: It is, checking the communication between the Sub Module to the Main odule.

river: In above example, assume that, if Main module A is under construction. At that me, the Developer take the help of a Temporary program (called as Driver) to check the ommunication between the modules.

2. What is the difference between black-box testing and white-box testing? **[MODEL QUESTION]**

nswer:

) Generally black box testing will begin early in the software development i.e. in requirement gathering phase it. But for white box testing approach one has to wait for the designing has to complete.

) We can use black testing strategy almost any size either it may be small or large. But white box testing will be effective only for smallness of codes or piece of codes.

) In white box testing we cannot test Performance of the application. But in Black box testing we can do it.

) In general we have to write large quantity of test cases for white box. But in black box its a selection of sample test cases.

3. What is meant by code walk-through? List the important types of errors hecked during code walk-through. **[MODEL QUESTION]**

Answer:

Code Walk-through: A formal testing technique where source code is traced by a group with a small set of test cases while the state of program variables is manually monitored o analyze the programmer's logic and assumptions

1. Flaws or potential flaws
2. Consistency with the overall program design
3. The quality of comments
4. Adherence to coding standards.
5. Existence, for the functionality implemented.
6. Clarity and Readability
7. Completeness - e.g., User, Training, Reference, Table Of Contents.

14. What is big bang approach of integration testing? **[MODEL QUESTION]**

Answer:

n big bang Integration testing, individual modules of the programs are not integrated ntil everything is ready.

This approach is seen mostly in inexperienced programmers who rely on 'Run it and see' approach.

15. What is stress testing? Why is stress testing applicable to only certain type systems? **[MODEL QUESTION]**

Answer:

Stress testing is a type of performance testing focused on determining an application's robustness, availability, and reliability under extreme conditions. The goal of stress testing is to identify application issues that arise or become apparent only under extreme conditions. These conditions can include heavy loads, high concurrency, or limited computational resources. Proper stress testing is useful in finding synchronization and timing bugs, interlock problems, priority problems, and resource loss bugs. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in an acceptable manner (e.g., not corrupting or losing data).

Stress tests typically involve simulating one or more key production scenarios under a variety of stressful conditions. For example, you might deploy your application on a server that is already running a processor-intensive application; in this way, your application is immediately "starved" of processor resources and must compete with the other application for processor cycles. You can also stress-test a single Web page or even a single item such as a stored procedure or class.

This chapter presents a high-level introduction to stress-testing a Web application. Stress testing can help you identify application issues that surface only under extreme conditions.

Examples of Stress Conditions

Examples of stress conditions include:

- Excessive volume in terms of either users or data; examples might include a denial of service (DoS) attack or a situation where a widely viewed news item prompts a large number of users to visit a Web site during a three-minute period.
- Resource reduction such as a disk drive failure.
- Unexpected sequencing.
- Unexpected outages/outage recovery.

Examples of Stress-Related Symptoms

Examples of stress-related symptoms include:

- Data is lost or corrupted.
- Resource utilization remains unacceptably high after the stress is removed.
- Application components fail to respond.
- Unhandled exceptions are presented to the end user

Long Answer Type Questions

1. a) What do you mean by software reliability? **[WBUT 2013, 2016]**
b) Briefly describe one reliability model. **[WBUT 2013]**

wer:
) Reliability is:
The probability of failure-free software operation for a specified period of time in a specified environment
Probability of the product working "correctly" over a given period of time.
Informally denotes a product's trustworthiness or dependability.
Software Reliability is an important attribute of software quality, together with
- functionality,
- usability,
- performance,
- serviceability,
- capability,
- installability,
- maintainability,
- documentation.

The Jelinski Moranda model is one of the most primitive reliability model. The unctional representation of the failure intensity can be expressed in the form
te of failure(t)= Φ(N-i+1) , where time interval of failure is assumed to be same
= Total number of errors present
= Number of errors found at time intervals
he set of observations for our case is:
N=50 errors
I=30 failure
Φ=0.03 (given)
te of failure(t)=0.03(50-30+1)=0.03x21 = .63 failure/cpu Hr.
After 40 failures
te of failure(t)=0.03(50-40+1)=0.03x11 = .33 failure/cpu Hr
Hence the rate of failures is decreasing with number of failures

2. Explain the LOC, Function point and Feature point. [WBUT 2014]

OR,

Why we use FP instead of LOC? Why do you think the FP need to be adjusted? [WBUT 2017]

Answer:

Line of Code (LOC)

The most commonly used measure of source code program length is the number of lines of code (LOC) (Fenton, 1997). The abbreviation NCLOC is used to represent a non-commented source line of code. NCLOC is also sometimes referred to as effective lines of code (ELOC). NCLOC is therefore a measure of the uncommented length.

The commented length is also a valid measure, depending on whether or not line documentation is considered to be a part of programming effort. The abbreviation CLOC is used to represent a commented source line of code (Fenton, 1997).

By measuring NCLOC and CLOC separately we can define:
total length (LOC) = NCLOC + CLOC
KLOC is used to denote thousands of lines of code.

Function points

Function points (FP) measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC). Counts are made for the following categories (Fenton, 1997):

*1. **External inputs*** – those items provided by the user that describe distinct application-oriented data (such as file names and menu selections).

*2. **External outputs*** – those items provided to the user that generate distinct application-oriented data (such as reports and messages, rather than the individual components of these)

*3. **External inquiries*** – interactive inputs requiring a response

*4. **External files*** – machine-readable interfaces to other systems

*5. **Internal files*** – logical master files in the system

Once this data has been collected, a complexity rating is associated with each count according to Table 1.

	Weighting Factor		
Item	**Simple**	**Average**	**Complex**
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
External files	7	10	15
Internal files	5	7	10

Table 1: Function point complexity weights

Each count is multiplied by its corresponding complexity weight and the results are summed to provide the UFC. The *adjusted function point count* (FP) is calculated by multiplying the UFC by a *technical complexity factor* (TCF). Components of the TCF are listed in Table 2.

F1	Reliable back-up and recovery	F2	Data communications
F3	Distributed functions	F4	Performance
F5	Heavily used configuration	F6	Online data entry
F7	Operational ease	F8	Online update
F9	Complex interface	F10	Complex processing
F11	Reusability	F12	Installation ease
F13	Multiple sites	F14	Facilitate change

Table 2: Components of the technical complexity factor

Each component is rated from 0 to 5, where 0 means the component has no influence on the system and 5 means the component is essential. The TCF can then be calculated as:
$TCF = 0.65 + 0.01(SUM(Fi))$

factor varies from 0.65 (if each *Fi* is set to 0) to 1.35 (if each *Fi* is set to 5) (Fenton,). The final function point calculation is:

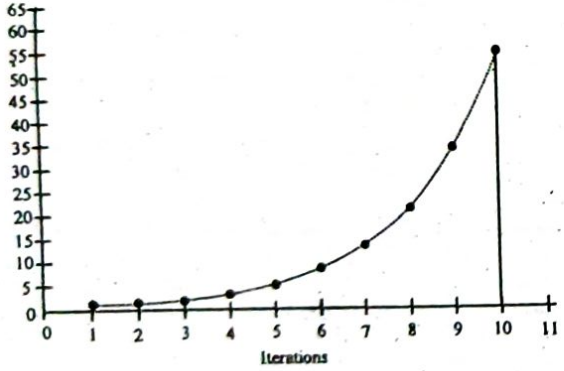
$UFC \times TCF$

ure point

/ points are an estimate of how long it will take to develop a user story/project. Story s represent time. Time is our key to project execution, clients and customers care t. They only care about complexity to the extent it influences the amount of time thing will take. It is an estimate of the effort involved can be influenced by risk, rtainty, and complexity.

/ point is a arbitrary measure used to measure the effort required to implement a . In simple terms its a number that tells the team how hard the story is. Hard could lated to complexity, Unknowns and effort. In most cases a story point range is 1, 2, 6 or X Small, Small, Medium, Large, Extra Large.

tly commonly used series is the fibonacci series. A fibonacci sequence is 3,5,8,13,21,34,45. Teams use a modified version of this which looks like 3,5,13,40,100. The reason is because the original sequence suggests mathematical iracy and real projects are not like that.



a relative term and does not directly co relate to actual hours. Since story points have elevance to actual hours, it makes it easy to think abstract about the effort required to iplete a story.

If you look at the Fibonacci curve it really takes a steep climb. If using this series consider not using 1 and 2. Use 3, 5, 8, 13, 40,100.

But how do you know which story is a 3 and which is a five. In order to do that each team would have to find a baseline story. It does not have to be the smallest one, but one that all in the team can relate too. From then on all sizing should be done compared to that baseline. This also creates a lot of confusion as story points do not relate to hours. So let's just not compare them. There is another technique called ideal hours which can be used.

Story points create lots of vagueness to your agile process. For every team, story size could mean different things depending on what baseline they chose. IF two teams are given exactly the same stories one can say their velocity is 46 and the other can say 14. Depends on what numbers they chose.

So if you want to compare velocity between teams that's a really bad idea as comparing velocity is like comparing apples and oranges. So do not compare velocity across teams. If you really have to track time then don't use story points at all,

3. a) What is meant by a Stub? What is a Driver?
b) With some suitable examples, explain statement coverage, branch coverage, and path coverage criteria.
c) Design a white box test suite for the following piece of C code:

```
int binary_search (int num)
{   int min, max;
        min = 0;
        max = 100;
        while (min! = max) {
            if (arr[(min + max)/2] > num)
                max = (min + max)/2;
            else if(arr[(min + max)/2] < num)
                min = (min + max)/2
            }
            return (–1);
}
```

The suite should include Control Flow Graph, Independent Path, and Cyclomatic Complexity (using two different techniques). **[WBUT 2015]**

Answer:

a) 1st Part:

A Stub is a dummy procedure, module or unit that stands in for an unfinished portion of a system.

Four basic types of Stubs for Top-Down Testing are:

1 Display a trace message
2 Display parameter value(s)
3 Return a value from a table
4 Return table value selected by parameter

stub is a computer program which is used as a substitute for the body of a software ıo e that is or will be defined elsewhere or a dummy component or object used to mulate the behavior of a real component until that component has been developed.

nd **Part:**

r is a the piece of code that passes test cases to another piece of code. Test Harness a test driver is supporting code and data used to provide an environment for testing f a system in isolation. It can be called as a software module which is used to ıvoke a module under test and provide test inputs, control and, monitor execution, and test results or most simplistically a line of code that calls a method and passes that ethod a value.

) For **Statement Coverage** and **Branch Coverage**

efer to Question No. 4(a) of Long Answer Type Questions.

ath Coverage

ach independent path in the code is taken in a predetermined order. Each statement in e software is executed at least once.

et us assume that a system is made up of only one program and that the program is only single loop.

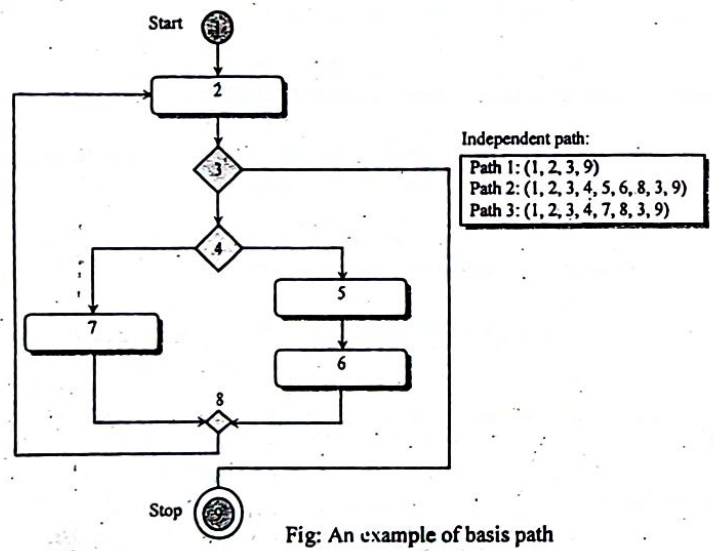
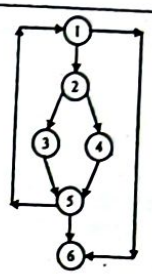


Fig: An example of basis path

1	int binary_search (int num)
	{int min, max; min = 0;max = 100;
2	while (min! = max) {

3	if (arr[(min + max)/2] > num) min = (min + max)/2
4	else if(arr[(min + max)/2] < num) min = (min + max)/2}
5	return (–1);
6	}



Control Flow Graph

Independent Paths are:

a) 1-6 while (min! = max) { [false]
b) 1-2-3-5-6
c) 1-2-4-5-6
d) 1-2-3-5-1-6 (min! = max) [True]
e) 1-2-4-5-1-6 (min! = max) [False]

Cyclomatic Complexity

a) V(G) = E – N + 2 = 8 – 6 + 2 = 4
b) V(G) = Total Number of bounded Areas + 1
= 3 + 1
c) V(G) = Total Number of decision points + 1
= 3 + 1 = 4

4. a) Discuss the limitations of testing. Why do we say that complete testing is impossible?
b) What are the various kinds of function testing? Describe any one in detail.
c) Explain the boundary value analysis testing technique with the help of an example.
[WBUT 2016]

Answer:

a) Testing has a number of pitfalls that make it far less effective and efficient than it should be. Testing is relatively ineffective in the sense that a significant number of residual defects remain in a completed system when it is placed into operation. Testing is also relatively inefficient when you consider the large amount of effort, funding, and time that is currently spent to find defects. Some of the significant testing limitations are tested below:

1. A program cannot be tested completely
2. Testing can be performed against specific system requirements
3. You cannot test every path
4. You cannot test every valid input

5. You cannot test every invalid input
6. Incomplete or ambiguous requirements may lead to inadequate or incorrect testing.
7. Exhaustive (total) testing is impossible at any present scenario.
8. Time and budget constraints normally require very careful planning of the testing effort.
9. In correct test results lead to make wrong business decisions.
10. Even if you do find the last bug, you'll never know it is the end of it.
11. You may run out of time before you run out of test cases.

We say complete testing is impossible because of the following reasons:

The realm of likely inputs of a program is often hefty to be fully used in testing a system. There are both valid inputs and invalid inputs. The program may have a large number of states. There may be timing constraints on the inputs, that is, an input may be valid at a certain time and invalid at other times. An input value which is valid but is not properly timed is called an inopportune input. The input domain of a system can also be huge to be entirely used in testing a program.

- The design issues may be too complex to completely test. The design may have included implicit design decisions and assumptions. For example, a programmer may use a global variable or a static variable to control program execution.
- It may not be possible to create all possible execution environments of the system. This becomes more significant when the behaviour of the software system depends on the real, outside world, such as weather, temperature, altitude, pressure, and so on.

b) The techniques used for functional testing are often specification-based. Functional Testing is the type of testing done against the business requirements of application. It is a black box type of testing and it is not concerned about the source code of the application.

Each and every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results. This testing involves checking of user interface, APIs, Database, security, client/ server applications and functionality of the application under test. The testing can be done either manually or using automation. In functional testing we check the functionalities of the software system. It mainly concentrates on -

1. ***Mainline functions:*** Testing the main functions of an application
2. ***Basic Usability:*** It involves basic usability testing of the system. It checks Whether a user can freely navigate through the screens without any difficulties.
3. ***Accessibility:*** Checks the accessibility of the system for the user
4. ***Error Conditions:*** Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

A comprehensive list of functional testing are:

a) Unit Testing
b) Smoke Testing
c) Sanity Testing
d) Integration Testing
e) White box testing

f) Black Box testing
g) User Acceptance testing
h) Regression Testing

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. Unit Testing is the first level of testing and is performed prior to Integration Testing. It is performed by software developers themselves or their peers. In rare cases it may also be performed by independent software testers.

Some of the benefits of Unit Testing are:

1. Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change.
2. Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
3. The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.
4. Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/ weeks/ months need to be scanned.

c) Boundary value analysis is another black box test design technique and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input. BVA is based on testing at the boundaries between partitions.

Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions). Test cases are designed based on the both valid and invalid boundary values. Typically, we choose one test case from each boundary.

As an example, consider a printer that has an input option of the number of copies to be made, from 1 to 99. To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case).

One test case for exact boundary values of input domains each means 1 and 100.

One test case for just below boundary value of input domains each means 0 and 99.

One test case for just above boundary values of input domains each means 2 and 101.

In this example we would have three equivalence partitioning tests (one from each of the three partitions) and four boundary value tests.

5. Define module coupling and explain different types of coupling. [WBUT 2016]

Answer:

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

Coupling can be "low" or "high". Some types of coupling, in order of highest to lowest upling, are as follows:

Content coupling (high)

ontent coupling occurs when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module). In this situation, a change the way the second module produces data (location, type, timing) might also require a change in the dependent module.

Common coupling

Common coupling (also known as **Global coupling**) occurs when two modules share the same global data (e.g., a global variable). Changing the shared resource might imply changing all the modules using it.

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

ontrol 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).

tamp coupling (Data-structured coupling)

Stamp coupling occurs when modules share a composite data structure and use only parts it, possibly different parts (e.g., passing a whole record to a function that only needs one field of it).

this situation, a modification in a field that a module does not need may lead to changing the way the module reads the record.

ata 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).

Message coupling (low)

This is the loosest type of coupling. It can be achieved by state decentralization (as in objects) and component communication is done via parameters or message passing.

No coupling

Modules do not communicate at all with one another.

6. a) What is software failure? How is it related with fault? **[WBUT 2016]**

Answer:

A **software failure** is a deviation **between** the specified and the actual behavior. In other words, the **software** does not do what the requirements describe. More precisely it is the inability of a system or component to perform required function according to its specification.

Failures may also arise because of human error in interacting with the software, perhaps a wrong input value is being entered or an output being misinterpreted. Finally failures may also be caused by someone deliberately trying to cause a failure in the system. Some of the major causes of failure may be described as follows:

- **Hardware failure:** Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.
- **Software failure:** Software fails due to errors in its specification, design or implementation.
- **Operational failure:** Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.

A **software fault**, or more commonly a "bug", is a mistake that may cause a **failure**. It is a condition that causes the **software** to fail to perform its required function. Failures are a usually a result of system errors that are derived from faults in the system. However, faults do not necessarily result in system errors if the erroneous system state is transient and can be 'corrected' before an error arises. Errors do not necessarily lead to system failures if the error is corrected by built-in error detection and recovery mechanism. In order to achieve the reliability the faults can be classified as follows:

- **Fault avoidance:** Development technique are used that either minimize the possibility of mistakes or trap mistakes before they result in the introduction of system faults.
- **Fault detection and removal:** Verification and validation techniques that increase the probability of detecting and correcting errors before the system goes into service are used.
- **Fault tolerance:** Run-time techniques are used to ensure that system faults do not result in system errors and/or that system errors do not lead to system failures.

b) Explain the Boehm software quality model with the help of a block diagram. **[WBUT 2016]**

Answer:

Boehm model was introduced in the year of 1978. The model is used to represent a hierarchical model that structures around high level characteristics, intermediate level characteristics, and primitive characteristics. All of these together results in to establishment of a high quality software model.

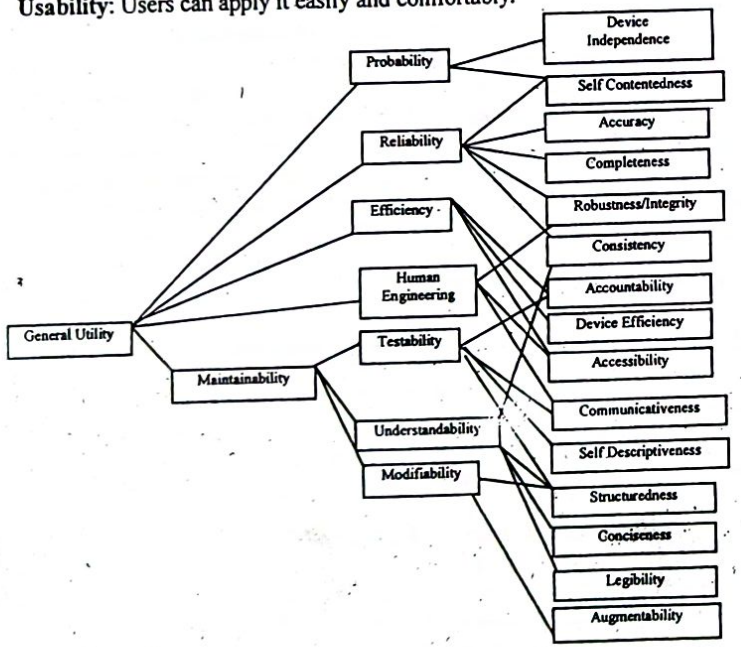
The model defines the quality of software on the basis of a set of credentials and measurements. It is also elucidates a model of software quality characteristics. The high level of characteristics is made in such a way that answers following questions:

- As-is utility: It defines the way a utility signifies the as-is utility. It creates a question of how easily, reliably and efficiently an as can be utilized.

- Maintainability: This aspect decides how convenient it is to understand, change or re-evaluate a process.
- Portability: This aspect helps in deciding an effective way to change an environment.

The intermediate level of characteristics represented by the model displays seven quality factors that altogether signify expected quality from a software system. These are as follows:

- **Flexibility**: It is very easy to amend the software as per the requirement. Parameters of the software should be so flexible that they can react on numerous situations.
- **Reliability**: Software performance should be reliable with zero defects. Result should be accurate.
- **Portability**: Software can run on different computer' program example DOS, windows.
- **Efficiency**: Practical & efficient use of resources or data collected. Optimum utilization of resources should be made.
- **Testability**: Software should be tested easily and as a result users can easily check that the results are correct so that they can rely on result blindly.
- **Understandability**: Software should be simple to understand for users so that they can use it properly and efficiently.
- **Usability**: Users can apply it easily and comfortably.



The factors can result into creation of other measurable properties. These are as follows:

1. Device Independence
2. Accuracy
3. Completeness
4. Robustness
5. Consistency
6. Accountability
7. Device efficiency
8. Accessibility
9. Communicativeness
10. Self-descriptiveness
11. Legibility
12. Augment ability

The Boehm Model was created after doing improvements in McCall software quality model encountered few errors in it. This new model of Boehm was found to be more interesting as it is placed in hierarchical order. The order begins with addressing the major concerns of the end-users. On the contrary, the bottom of the hierarchy displays the technically inclined personnel.

The model focuses on measuring properties and characteristics in such a way that create complex and non-technical stakeholders that are involved in the lifecycle of software. As compared to McCall model, this model is used in a widespread manner because of its bottom to top approach of software quality.

Even though this software quality model overcomes the shortcomings of various olden software quality models and as it provides a basic amount of support by following a top down approach to quality of software, this model is also short-lived as far as a solid software quality testing is expected to be.

7. Write short notes on the following:

a) Statement coverage vs. branch coverage **[WBUT 2013]**
b) Verification and Validation **[WBUT 2013, 2015]**
c) Six Sigma **[WBUT 2014]**
d) ISO vs. CMM standards **[WBUT 2014]**
e) Software quality metrics **[WBUT 2014]**
f) Function point method **[WBUT 2015]**
g) Cyclomatic complexity **[WBUT 2016]**
h) Software Reengineering **[WBUT 2016]**
i) Black box testing **[WBUT 2017]**
j) Quality Assurance **[WBUT 2018]**
k) Regression Testing **[WBUT 2018]**

Answer:

a) Statement coverage vs. branch coverage:

Let's consider following piece of a code:

public int returnInput(int input, boolean condition1, boolean condition2, boolean condition3)

```
 input;
t y = 0;
ondition1)
+;
(condition2)
;
(condition3)
turn y;
```

atement coverage

order to execute every statement we need only one testcase which would set all itions to true, every line of a code (statement) is touched.

urnInput(x, true, true, true) - 100% statement covered But only half of branches are d and only one path.

anch coverage

n visualize every "if-statement" as two branches (true-branch and false-branch). it can clearly be seen that the above testcase follows only "true-branches" of every "if ent". Only 50% of branches are covered.

order to cover 100% of branches we would need to add following testcase: put (x, false, false, false)

th these two testcases we have 100% statements covered, 100% branches covered

Verification and Validation:

Refer to Question No. 2 of Short Answer Type Questions.

Sigma:

a business management strategy that was initially developed by Motorola in the 0's, and now is used in many fortune 500 companies. It is used primarily to identify rectify errors and defect in a manufacturing or business process. Six Sigma uses a ber of quality methods and tools that are used by professionals within the anization, who have been trained on Six Sigma techniques.

: Six Sigma DMAIC process (defines, measure, analyze, improve, control) is an ement system for existing processes falling below specification and looking for emental improvement. The Six Sigma DMADV process (define, measure, analyze, , verify) is an improvement system used to develop new processes or products at Sigma quality levels. It can also be employed if a current process requires more than cremental improvement. Both Six Sigma processes are executed by Six Sigma en Belts and Six Sigma Black Belts, and are overseen by Six Sigma Master Black

d) ISO vs. CMM standards:

An international standard which provides broad guidance to software developers on how to Implement, maintain and improve a quality software system capable of ensuring high quality software consists of 20 requirements. It differs from country to country.

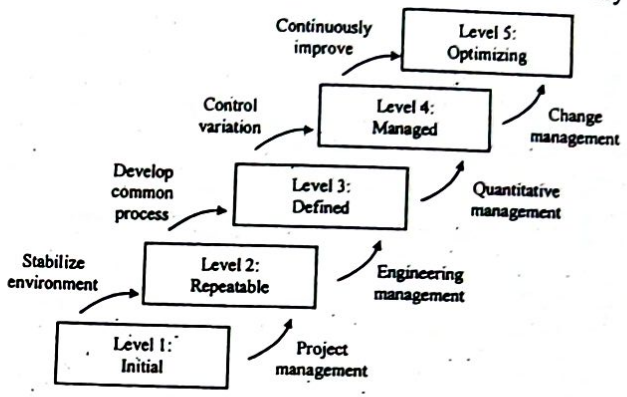


Fig: Relation between ISO and CMM

1) ISO 9000 describes quality assurance elements in generic terms that can be applied to any business.
2) It treats an enterprise as a network of interconnected processes.
3) To be ISO-complaint processes should adhere to the standards described.
4) Elements include organizational structure, procedures, processes and resources.
5) Ensures quality planning, quality control, quality assurance and quality improvement.

CMM: The CMM model was developed by SET (Software Engineering Institute) of Carnegie-Mellon University in 1986. The model is a strategy for improving the software process, irrespective of the actual life cycle model used. It is used to judge the maturity of the software processes of an organization and to identify the key practices. The CMM is organized into five maturity levels as shown in figure above. The five maturity models are summarized as follows:

There are no key process areas for *level 1.*

The key process areas at *level 2* focus on the software project's concerns related to establishing basic project management controls, as summarized below:

1. Requirements Management (RM)
2. Software Project Planning (PP)
3. Software project Tracing and Oversight (PT).
4. Software Subcontract Management (SM)
5. Software Quality Assurance (QA)
6. Software Configuration Management (CM)

The key process areas at *level 3* address both project and organizational issues as summarized below:

1. Organization Process Focus (PF)
2. Organization Process Definition (PD)
3. Training Program (TP)
4. Integrated Software Management (IM)
5. Software Product Engineering (PE)
6. Inter group Coordination (IC)
7. Peer Reviews (PR)

The key process areas of *level 4* focuses on establishing a quantitative understanding of both the software process and the software work products as summarized below:

1. Quantitative Process Control Management (QP)
2. Software Quality Management (QM)

The key process areas of *level 5* cover the issues that both the organization and the projects as summarized below:

1. Defect Prevention (DP)
2. Technology Change Management (TM)
3. Process Change Management (PC)

e) Software quality metrics:

Software is a guide to measuring software quality. It is a quantitative measure of a degree to which a software system or process holds some property. The objective is to bring 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. Some of the common software measurements include:

1. Code coverage
2. Cohesion
3. Coupling
4. Cyclomatic complexity (McCabe's complexity)
5. DSQI (design structure quality index)
6. Function Points and Automated Function Points
7. Halstead Complexity
8. Instruction path length
9. Maintainability index
10. Number of classes and interfaces
11. LOC
12. Program execution time
13. Program load time

Program size (binary)

f) Function point method:

Refer to Question No. 2 (2nd Part) of Long Answer Type Questions.

g) Cyclomatic complexity:

Cyclomatic complexity is a software metric (measurement), used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code.

McCabe's Cyclomatic complexity

Cyclomatic complexity introduced by Thomas McCabe in 1976. It simply measures the amount of decision logic in the program module. Cyclomatic complexity gives the minimum number of paths that can generate all possible paths through the module. Cyclomatic complexity is often referred to as McCabe's complexity. McCabe's complexity used to define minimum number of test cases required for a module and it is used during software development lifecycle to quantify maintainability and testability.

Cyclomatic complexity is defined as

$CC = E - N + P$

Where

E = the number of edges of the graph

N = the number of nodes of the graph

P = the number of connected components

In case of connected graph

$CC = E - N + 2$

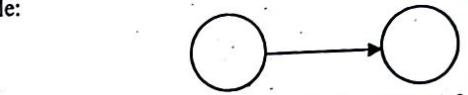
In simplified way it can be defined as

$CC = D + 1$

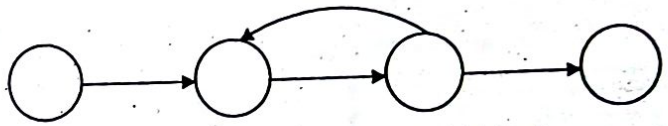
Where

D = the number of decision points in the graph

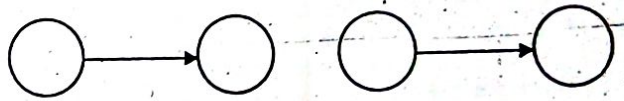
For example:



E = 1 N = 2 P = 1 CC = 1-2+2 = 1



E = 4 N = 4 P = 1 CC = 4-4+2 = 2



E = 2 N = 4 P = 2 CC = 2-4+2×2 = 2

h) Software Re-engineering:

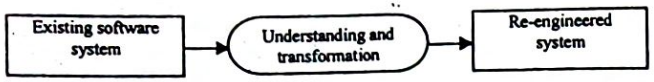
Restructuring or rewriting part or all of a system without changing its functionality. It is applicable when some (but not all) subsystems of a larger system require frequent maintenance.

involves putting in the effort to make it easier to maintain. The reengineered system may also be restructured and should be re-documented.

When do you decide to reengineer?

1. When system changes are confined to one subsystem, the subsystem needs to be reengineered
2. When hardware or software support becomes obsolete
3. When tools to support restructuring are readily available

The process diagram for Re-engineering



Software re-engineering

ngineering advantages

Reduced risk

There is a high risk in new software development. There may be development problems, staffing problems and specification problems

Reduced cost

The cost of re-engineering is often significantly less than the costs of developing new software

engineering cost factors

The quality of the software to be re-engineered
tool support available for re-engineering
The extent of the data conversion which is required
availability of expert staff for re-engineering
Thus the key points of Re-Engineering are:
verse engineering is the process of deriving the system design and specification from its source code.
rogram structure improvement replaces unstructured control constructs with while loops and simple conditions.
Program modularisation involves reorganisation to group related items
a re-engineering may be necessary because of inconsistent data management

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle. This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

There are different techniques involved in Black Box testing.

a. Equivalence Class
b. Boundary Value Analysis
c. Domain Tests

d. Orthogonal Arrays
e. Decision Tables
f. State Models
g. Exploratory Testing
h. All-pairs testing

Types of Black Box Testing

There are many types of Black Box Testing but following are the prominent ones -

Functional testing - This black box testing type is related to functional requirements of a system; it is done by software testers.

Non-functional testing - This type of black box testing is not related to testing of a specific functionality, but non-functional requirements such as performance, scalability, usability.

Regression testing - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Black box testing strategy:

Following are the prominent Test Strategy amongst the many used in Black box Testing

Equivalence Class Testing: It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.

Boundary Value Testing: Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is mostly suitable for the systems where input is within certain ranges.

Decision Table Testing: A decision table puts causes and their effects in a matrix. There is unique combination in each column.

j) Quality Assurance

Quality Assurance Factors:

Utility/Correctness: extent to which program satisfies specifications (presence of function)
Reliability: precision with which program performs as expected (absence of failure)
Modifiability: effort required to modify
Maintainability: effort required to locate and fix errors
Security: extent to which access is controlled
Efficiency: amount of computing resources required
Usability: effort required to learn, operate, and interpret behavior
Testability: effort required to test
Interoperability: effort required to couple to another system

Portability: effort required to transfer to different environment
Reusability: extent to which program can be used in other applications.

k) Regression Testing:
Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process and, in larger companies, is done by code testing specialists. Test department coders develop code test scenarios and exercises that will test new units of code after they have been written. These test cases form what becomes the test bucket. Before a new version of a software product is released, the old test cases are run against the new version to make sure that all the old capabilities still work. The reason they might not work is because changing or adding new code to a program can easily introduce errors into code that is not intended to be changed.

8. a) What is the difference between a coding standard and coding guideline? Why are these considered important in a software development organization? [MODEL QUESTION]

Answer:
Coding guideline:
A rule about how you are to write your code so that it will be consistent, easily understood and robust.

Coding standard:
A rule about how you are to write your code so that it will be consistent, easily understood and robust and so that it will be acceptable to the entity that is paying you to write the code.
Though at initial stages it seems like a burden to follow standard, its advantages become visible once the software grows to few thousand lines spanning few hundred files. Some of the advantages are:
Programmer feels comfortable with the code written by others, as it is similar to what he himself would have written.
Person joining the group at later stage can pickup the code easily (once he is familiar with the standards).

b) Why are the three different levels of testing — unit testing, integration testing and system testing — is necessary to the software development? What is the main purpose of each of these different levels of testing? [MODEL QUESTION]

Answer:
In computer programming, unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. Unit tests are created by programmers or occasionally by white box testers.
Ideally, each test case is independent from the others: substitutes like method stubs, mock objects,[1] fakes and test harnesses can be used to assist testing a module in isolation.

Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. Its implementation can vary from being very manual (pencil and paper) to being formalized as part of build automation.

Integration testing (sometimes called Integration and Testing, abbreviated "I&T") is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

9. Design a test suite for testing your binary search function. [MODEL QUESTION]

Answer:
Path:
1. 1-11
2. 1-2-3-10
3. 1-2-4-5-9-10-11
4. 1-2-4-6-7-8-9-10-11
5. 1-2-4-6-8-9-10-11

Path 1-11
Test case: (low>high)

Path: 1-2-3-10
(low=high) and (key < b[mid])
(low<high) and (key < b[mid])

Path: 1-2-4-5-9-10-11
(low=high) and (Key=b[mid]) and (Key >b[mid])
(low=high) and (Key>b[mid]) and (Key >b[mid])
(low<high) and (Key=b[mid]) and (Key >b[mid])
(low<high) and (Key>b[mid]) and (Key >b[mid])

Path: 1-2-4-6-7-8-9-10-11
(low=high) and (Key=b[mid]) and (Key <b[mid]) and (b[mid]==key)
(low=high) and (Key=b[mid]) and (Key =b[mid]) and (b[mid]==key)
(low=high) and (Key>b[mid]) and (Key =b[mid]) and (b[mid]==key)

10. a) What do you understand by the term 'system testing'? What are the different kinds of system testing that are usually performed on large software products? [MODEL QUESTION]

Answer:
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

ing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.
a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself grated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together lled assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-emblages" and also within the system as a whole.
The following examples are different types of testing that should be considered during tem testing:

- GUI software testing
- Usability testing
- Performance testing
- Compatibility testing
- Error handling testing
- Load testing
- Volume testing
- Stress testing
- Security testing
- Scalability testing
- Sanity testing
- Smoke testing
- Exploratory testing
- Ad hoc testing
- Regression testing
- Reliability testing
- Installation testing
- Maintenance testing
- Recovery testing and failover testing.

b) What is the difference between the top-down and the bottom-up integration testing approaches? [MODEL QUESTION]

Answer:

Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.
the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of odules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This thod also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.
Sandwich Testing is an approach to combine top down testing with bottom up testing.
The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link.

11. a) What is unit testing & what errors are found during unit testing?
b) Describe McCall's Quality Assurance Factors.
c) What is symbolic execution? Consider the following function:

```
function product (x, y, z: integer) : integer;
    var t1, t2 : integer;
    var prod : integer;
    begin
      t1 : = x * y;
      t2 : = y * x;
      prod : = t1 * t2/y;
    end;
```

Give the symbolic execution of the above function. [MODEL QUESTION]

Answer:

a) ***Unit Testing:***

In computer programming, a **unit test** is a procedure used to validate that a particular module of source code is working properly. The procedure is to write test cases for all functions and methods so that whenever a change causes a regression, it can be quickly identified and fixed. Ideally, each test case is separate from the others; constructs such as mock objects can assist in separating unit tests. This type of testing is mostly done by the developers and not by end-users.

Advantage of unit testing:

- Unit tests can provide quick feedback to the developer.
- Unit tests can simplify the structure of the system.
- Unit tests can mitigate concerns about the effects of refactoring.
- Unit tests can be used to validate code integration.
- Unit tests may result in more testable code.
- Unit tests can document the code they test.
- Unit tests are typically inexpensive to run and maintain.
- Unit tests report serious problems early.

b) ***Software Quality Factors/Goals:***
Utility/Correctness: extent to which program satisfies specifications (presence of function)
Reliability: precision with which program performs as expected (absence of failure)
Modifiability: effort required to modify
Maintainability: effort required to locate and fix errors

Security: extent to which access is controlled
Efficiency: **amount** of computing resources required
Usability: effort required to learn, operate, and interpret behavior
Testability: effort required to test
Interoperability: effort required to couple to another system
Portability: effort required to transfer to different environment
Reusability: extent to which program can be used in other applications

c) In computer science, symbolic execution (also symbolic evaluation) refers to the analysis of programs by tracking symbolic rather than actual values, a case of abstract interpretation. The field of symbolic simulation applies the same concept to hardware. Symbolic computation applies the concept to the analysis of mathematical expressions. Symbolic execution is used to reason about all the inputs that take the same path through a program.

QUESTION 2014

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following:

i) In Integration Testing approach, where all modules making up a system are integrated in a single step is known as –
a) top-down Integration Testing
✓b) bottom-up Integration Testing
c) big-band Integration Testing
d) mixed Integration Testing

ii) MTTF is a measure of –
✓a) reliability
b) maintainability
c) cost of effort
d) testability

iii) To allocate resource to activities we use
a) PERT chart
b) Gnatt chart
c) Network Diagram
✓d) all of these

iv) To achieve a good design, modules should have –
a) weak cohesion low coupling
b) weak cohesion high coupling
✓c) strong cohesion low coupling
d) strong cohesion high coupling

v) Which is NOT a non-functional requirement?
a) efficiency
b) reliability
✓c) product features
d) stability

vi) If data from one module is used to direct the order of execution in another, then the coupling is known as –
a) Stamp Coupling
b) Data Coupling
✓c) Control Coupling
d) Content Coupling

vii) Cardinality in an ER Diagram refers to
a) number of attributes in an entity
✓b) the order of co-related entities
c) the number of sub-entities
d) degree of a relationship

viii) Alpha-testing is done by
✓a) the development team
b) a friendly set of customers
c) the customer himself
d) none of these

ix) Which model is generally used for developing GUI of a system?
✓a) spiral
b) prototyping
c) iterative waterfall
d) evolutionary

x) Data hiding can be achieved by
✓a) Data Encapsulation
b) Data Overloading
c) Data Abstraction
d) Polymorphism

Group – B
(Short Answer Type Questions)

hat is Mutation Testing? Distinguish between White-box Testing and Black-box testing?
opic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 1.

n ER diagram for Hospital Management System showing cardinalities, strong and weak es, derived attributes, primary key etc.
opic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 2.

stinguish between software verification and software validation. When during the life cycle ation and validation performed?
opic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 2.

t are 'baselines' with respect to software configuration management?
hat is the necessity of software configuration management in developing a software?
opic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 6.

project was estimated to be 200 KLOC. Calculate the effort development time, average staff productivity level for
anic
etached modes.
opic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 7.

Group – C
(Long Answer Type Questions)

ain "Use Case" diagram. What are the essential criteria for ideal use case diagram? What he "extends" and "includes" constructs in use case diagram? Draw a use case diagram for Home functionality where examples of actors are Patient, Doctor, Reception Staff, Billing and Administrator etc.
plain Sequence and Activity diagram with example.
: SOFTWARE DESIGN & UML, Long Answer Type Question No. 3.

plain when and why you will use PERT charts and when and why you will use Gantt charts you are a project manager.
nsider a software project with 5 activities T1 to T5. Duration of 5 activities in weeks are 3, 2, 3, ectively. T2 and T4 can start when T1 is complete. T3 can start when T2 is complete. T5 tart when both T3 and T4 are complete. Draw activity network for the project. When is the tart date of the activity T3? What is the float of the activity T4? Which activities are on the path? Draw the Gantt chart also.
pic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 4.

Draw the context diagram and Level-1 DFD for Library management system. Draw also USE-gram for this system.
hat do you mean by balancing of DFD? Explain with a suitable example.
plain the LOC, Function point and Feature point.
) ee Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question

pic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 2.

What is SRS? Write the features of SRS.
is Risk? Why Risk Analysis is done?

c) What is "Top-Down and Bottom-Up Design" approach?
a) & b) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 4.
c) See Topic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 3.

11. Write short notes on any three of the following:
a) Software Configuration Management
b) Six Sigma
c) Decision Tree and Decision Table
d) ISO vs. CMM standards
e) Software quality metrics

a) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(b).
b) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(c).
c) See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 13(c).
d) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(d).
e) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(e).

QUESTION 2015

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following:
i) One way to improve readability in coding is to
✓a) avoid goto statements
b) name variables and functions according to their use
c) modularize the program
d) none of these

ii) The type of failure that occurs for all input values while invoking a function of the system is
a) transient failure ✓b) permanent failure
c) recoverable failure d) unrecoverable failure

iii) According to COCOMO number of cost drivers is
a) 10 ✓b) 15 c) 20 d) 14

iv) The bet type of cohesion is
a) coincidental b) logical c) informational ✓d) functional

v) A physical DFD specifies
a) what processes will be used b) who generates data and who processes it
c) what each person in an organization does ✓d) none of these

vi) Function point describes
a) the SRS document
b) the test plans
c) the functional decomposition
✓d) the size of a software product directly from its specification

Efforts are measured in terms of
✓a) Person-months b) Persons c) Rupees d) Months

Which form of software development model is most suited to a system where all the uirements are known at the start of a project and remain stable throughout the project?
✓a) Waterfall model b) Incremental model
c) Evolution model d) Spiral model

Which is not a size measure for software?
a) LOC b) Function count
✓c) Cyclomatic complexity d) Halstead's program length

he cyclomatic complexity of the following program fragment is:

```
in: gcd (int x, int y)
while (x! = y)
if (x > y) then
x = x – y;
else y = y – x;
}
return x;
}
```

a) 2 ✓b) 3 c) 4 d) 5

Group – B
(Short Answer Type Questions)

Vhat are Cohesion and Coupling? Mention different kinds of cohesion.
Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 3.

Vhat is Use Case diagram? Draw the Use Case diagram of Hospital Management System.
Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 4.

) Define software quality.
t is estimated that there will be 70 errors in a software. During testing 25 errors have been erienced. Calculate the failure intensity with a given value of $\phi = 0.03$ using Jelenski Moranda del. What will be the failure intensity after experiencing 50 errors?
ee Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 1(a).
ee Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 5.

Vhat are different levels of Testing and their goals?
Topic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 4.

he size of an organic type software product has been estimated to be 1,00,000 lines of source e. The average salary of software developers is Rs. 10,000/- per month. Determine the effort uired to develop software product, the nominal development time and the cost to develop the duct.
Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 8.

Group – C
(Long Answer Type Questions)

7. a) What is meant by a Stub? What is a Driver?
b) With some suitable examples, explain statement coverage, branch coverage, and path coverage criteria.
c) Design a white box test suite for the following piece of C code:

```
int binary_search (int num)
{  int min, max;
   min = 0;
   max = 100;
   while (min! = max) {
      if (arr[(min + max)/2] > num)
         max = (min + max)/2;
      else if(arr[(min + max)/2] < num)
         min = (min + max)/2
      }
      return (–1);
}
```

The suite should include Control Flow Graph, Independent Path, and Cyclomatic Complexity (using two different techniques).
See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 3.

8. a) Write down three advantages of decision trees over decision table.
b) Mention two situations where decision tables work best.
c) A bank has decided to adopt the following policy on deposits:
On deposit of Rs. 5,000/- and above and for three years or above the interest is 10%. On the same deposit for a period less than 3 years it is 8%. On deposits below Rs. 5,000 the interest is 6% regardless of the period of deposit.
Develop a decision tree and a decision table for the above process. Also express the above policy using structured English.
d) Distinguish between physical DFD and logical DFD with example of each.
See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 5.

9. a) Explain top-down and bottom-up design.
b) Distinguish between object oriented design and function oriented design with proper examples.
c) Explain the phase of Spiral model with advantages and disadvantages.
d) Explain the advantages and disadvantages of prototype model.
a) See Topic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 3.
b) See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 7.
c) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 9(a).
d) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 9(b).

10. Draw a context diagram and top level DFD for the following system. Also provide relevant data dictionary,
The admission committee determines whether or not a student is admitted to graduate school. The associated dean determines the financial aid.
Students send applications to the graduate school. From other universities, the graduate school receives transcripts. Additionally, unrelated third parties provide letter of recommendation. The Graduate Management Admissions Council (GMAC) provides GMAT scores. Upon receipt of the

above items, the graduate school prepares an application packet and enters the student's name in the pending application file. An acknowledgement letter is sent to the student. The graduate school sends the application packet to the admission committee. The admission committee reviews the student's credentials. In most cases the student's are accepted. The committee sends the applicant a letter explaining the result of review. A copy of the letter is also sent to the registrar who creates a student record in the registrar's system. The accepted student's file is sent to the associated dean for financial aid review. The associate dean keeps a list of available scholarships. Based on the review of the student's interests and the scholarship criteria a financial aid award letter is prepared, which again is sent to the student. Also, the controller's office receives a copy so that the proper bill can eventually be sent to the student once he/she registers for classes. The associate dean then updates the pending application file, closing out the student's record. Each month, the graduate school prepares a summary of how many applications have been received, approved and rejected. This report is sent to the university's president.

See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 6.

11. Write short notes on any three of the following:
   a) CASE tools
   b) Function point method
   c) Risk management
   d) Software configuration management
   e) Verification and validation.

See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(d).

See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(f).

c) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(c).

d) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(b).

See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(b).

QUESTION 2016

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for any *ten* of the following:

i) The most important feature of spiral model is
   a) Requirement analysis
   ✓c) Risk management
   b) Quality management
   d) Configuration management

ii) To achieve good design, modules should have
   a) Low coupling, low cohesion
   c) High coupling, low cohesion
   ✓b) Low coupling, high cohesion
   d) High coupling, high cohesion

iii) Equivalence class partitioning is followed in
   a) White box testing
   c) Both (a) and (b)
   ✓b) Black box testing
   d) None of these

iv) Project planning does not include
   a) Risk identification
   c) Cost estuation
   ✓b) Design
   d) Configuration Management

v) A COCOMO model is
   a) Common cost estimation model
   ✓c) Constructive cost estimation model
   b) Complete cost estimation model
   d) Comprehensive cost estimation model

vi) Each time a defect gets detected and fixed, the reliability of a software product
   a) decreases
   c) remains constant
   ✓b) increases
   d) cannot say anything

vii) All critical path activities have slack time of
   ✓a) 0
   b) 1
   c) 2
   d) None of these

viii) Alpha critical path activities have slack time of
   ✓a) Acceptance testing
   c) Integration testing
   b) System testing
   d) Unit testing

ix) In function point analysis the number of adjustment factors based on system characteristics to refine unadjusted function point is
   a) 12
   b) 10
   c) 20
   ✓d) 14

x) CASE Tool is
   ✓a) Computer Aided Software Engineering
   c) Constructive Aided Software Engineering
   b) Component Aided Software Engineering
   d) Computer Analysis Software Engineering

Group – B
(Short Answer Type Questions)

2. Discuss the characteristics of a good SRS document.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 10.

3. Explain in detail the Capability Maturity Model (CMM).
See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 2(d).

4. What is the difference between black-box and white-box testings?
See Topic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 1(2nd Part).

5. List three common types of risks that a typical software project might suffer from.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 11.

6. Draw the use case diagram of a library management system.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 12.

Group – C
(Long Answer Type Questions)

1) What is software engineering?
Discuss the software engineering process.
Explain waterfall model in detail with the help of diagram. State its advantage and also its tations.
Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 5.

3) Draw the E-R diagram for a hospital management.
What do you mean by the term 'requirement'? Explain the process of determining the uirements for a software based system.
Describe the various steps of requirements engineering. Is it essential to follow these steps?
See Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 2.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No.).
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No.

a) Discuss the limitations of testing. Why do we say that complete testing is impossible?
What are the various kinds of function testing? Describe any one in detail.
Explain the boundary value analysis testing technique with the help of an example.
e Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 4.

. a) Define module coupling and explain different types of coupling.
What is the difference between a flow chart and a structure chart?

List the points of a simplified design process.
See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 5.
& c) See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 8.

a) What is software failure? How is it related with fault?
What is software reliability?
Explain the Boehm software quality model with the help of a block diagram.
See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 6(a).
See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 1(a).
See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 6(b).

2. Write short notes on any three of the following:
a) Water fall Model
b) Cyclomatic Complexity
c) COCOMO Model
d) Software Re-engineering
e) DFD
) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 3(e).
) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(g).
) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 0(f).
) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(h).
) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 0(g).

QUESTION 2017

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following:

i) DFD shows
a) the flow of data
b) the processes
c) the areas where they are stored
✓d) all of these

ii) Coding and testing are done in which of the following manner?
✓a) Adhoc
b) Top-clown
c) Cross sectional
d) Bottom-up

iii) The largest percentage of total life cycle cost of software is
✓a) Design cost
b) Coding cost
c) Maintenance cost
d) Testing cost

iv) Software maintenance includes
a) Setting preventing maintenance policy for servers
b) Installation of software at site
c) Designing of software for maintenance purposes
✓d) Bug fixing

v) Main difference between program testing and system testing is
✓a) System testing focuses on testing the interfaces between programs, program testing focuses on individual programs
b) Program testing is more comprehensive than system testing
c) System testing is tough and program testing is easy
d) none of these

vi) The most creative and challenging phase of system life cycle is
✓a) Design
b) Feasibility study
c) Maintenance
d) none of these

vii) The database design activity deals with the design of
a) Logical database
b) Physical database
✓c) both (a) and (b)
d) none of these

viii) Coupling is a measure of
a) Relative functional strength
b) Interdependence among module
✓c) both (a) and (b)
d) none of these

ix) Decision support systems are used to
✓a) Evaluate and analyze the mission of organization
b) To perform accounting
c) Perform multiple activities simultaneously
d) none of these

ototype is a
a) Working model of existing system
b) Mini model of existing system
c) Mini model of processed system
d) none of these

Group – B
(Short Answer Type Questions)

xplain Spiral Model for software development with a diagram.
Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No.

Vhy Project Planning is needed? Draw the diagram for precedence ordering among planning
Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No.

xplain Empirical Cost Estimation Techniques.
ic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No.

What is meant by cohesion? How should software be designed considering cohesion? What is difference between cohesion and coupling?
ic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 6.

Distinguish between error and failure. Which of the two is detected by testing? Justify your
Topic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 5.

Group – C
(Long Answer Type Questions)

Suppose you are the Project Manager of a Software Project that consists of following Activities in table and you have to draw the activity network and find the critical tasks of the project.
he Gantt chart of the Project. (Consider Resources allocation will start from 12th March, 0).

Activity No.	Activity Name	Duration (weeks)	Immediate Predecessor
1.	Obtain Requirements	4	–
2.	Analyze Operations	4	–
3.	Define Subsystems	2	1
4.	Develop Database	4	1
5.	Make Decision Analysis	3	2
6.	Identify Constraints	2	5
7.	Build Module 1	8	3, 4, 6
8.	Build Module 2	12	3, 4, 6
9.	Build Module 3	18	3, 4, 6
10.	Write Report	10	6
11.	Integration and Testing	8	7, 8, 9
12.	Implementation	2	10, 11

Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 9.

Consider the following C program:

```
compute_gcd (x, y)
int x, y;
{
        while (x! = y)
                if (x > y) then x = x - y;
                else y = y - y;
        return x;
}
```

a) Find out the estimated length, Program vocabulary, Program volume, Effort, Time. Comment on the technique that you use to solve the problem.
b) Compare Halstead's length and volume measures of size with the LOC measure.
See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 10.

9. a) What is SDLCM? What are the Disadvantages in Classical Waterfall Model?
b) Why we use FP instead of LOC? Why do you think the FP need to be adjusted?
c) What is the Heuristic Project Estimation technique?
d) A Project size of 200 LOC is to be developed. Software development team has average experience on similar type of project. The project schedule is not very tight. Calculate Effort, Time of Development, Average Staff size and Productivity of the Project.
a) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 1.
b) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 2.
c) & d) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 6(a) & (b).

10. a) Draw a DFD that depicts and ATM system (only withdrawal) mentioning suitable Assumptions. Now build a structure chart.
b) An embedded system of size 400 KLOC has to develop. Project manager has a choice of hiring from two pools of developers: very highly capable with very little experience (AEXP very high) or developers of low quality (LEXP very low) but a lot of Experience. Find the values of EAF, Effort & Development Time.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 7.

11. Write short notes on any *three* of the following:
a) Team Structure of an Organization
b) Risk Identification & Assessment
c) CASE TOOLS
d) Black box testing
e) Work Break down Structure & Utility of PERT Chart
a) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(h).
b) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(i).
c) See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 10(d).
d) See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(i).
e) See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 13(d).

QUESTION 2018

Group – A
(Multiple Choice Type Questions)

Choose the correct alternatives of the following:

'racking the correspondence between the design component and the SRS is known as
a) availability ✓b) traceability c) maintainability d) reliability

Vhen the two bubbles are interconnected directly, it is referred as
a) serial DFD b) direct DFD ✓c) synchronous d) balanced DFD

DFD balancing means
a) balancing of weight of processes
b) must match the total number of bubbles
✓c) must match the data flow at the next level of DFD
d) None of the above

System Testing is performed by a friendly set of customer is known as
a) alpha testing ✓b) beta testing c) performance testing d) usability testing

:unction point describes
a) the SRS document b) the test plans
c) the functional decomposition ✓d) the size of a software product direct from its specification

The potential risks are best detected by
'a) spiral model b) waterfall model c) incremental model d) prototyping model

The most desirable form of cohesion is
a) sequential cohesion b) procedural cohesion
c) coincidental cohesion ✓d) functional cohesion

Software testing is the
a) process of demonstrating that errors are not present
b) process of establishing confidence that a program does what it is supposed to do
c) process of executing a program to show that it is working as per specifications
✓d) process of executing a program with the intent of finding errors

he best type of coupling is
a) coincidental b) logical c) informational ✓d) functional

g Bang Integration testing is useful for project with
) smaller number of modules b) large number of modules
) average number of modules ✓d) none of these

Group – B
(Short Answer Type Questions)

;piral model is not a panacea'; explain the meaning of the statement. Also state why spiral is
d a Meta-model.
:opic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 6.

3. What are the different of information elicitation?
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 15.

4. What are the metrics for estimation of software? State characteristics of feature point metrics.
1st Part: See Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 7.,
2nd Part: See Topic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 2 (3rd Part).

5. What are the major components of SRS?
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 18.

6. Develop a work breakdown structure specification for showing the process of admission to an engineering college. Assume major phase as exam preparation, entrance exam, admission criterion and counseling and fees payment. Also write the output of each major task performed.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 16.

Group – C
(Long Answer Type Questions)

7. What is cost benefit analysis? What are the common techniques for cost benefit analysis? Develop a set of functional and non-functional requirements for a new software project.
See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 8.

8. a) Consider a project with 950 lines of code which has the following distribution in terms of days for each effort:

Phase	Programmer Days
Requirements	20
Design	10
Implementation	10
Testing	15
Documentation	10

Calculate the productivity given the line of code and number of programmer days.
b) Calculate COCOMO effort, development time and productivity for an organic project that is estimated to have 39,800 lines of code. Assume values of constant a = 2.4 and b = 1.05.
c) Specify the general principles of user interface design.
a) & b) Part: See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No. 9(a) & (b).
c) See Topic: SOFTWARE DESIGN & UML, Short Answer Type Question No. 8.

9. a) What is Cohesion? Explain the cohesion classification with respect to software design. 'A good software should have high cohesion but low coupling' – Explain.
1st & 2nd Part: See Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 11.
3rd part: See Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Short Answer Type Question No. 17.

'av he context diagram and Level 1 and Level 2 DFD for the following correspondence course m. A college offers correspondence courses to students. Each course lasts 20 weeks and is o a weekly study module and progress test. At the end of the course students sit at an ilated examination. The college Registrar deals with enquiries and applications, and students who have sufficient qualifications are asked to register by completing and submitting an cation form. After approval by the Academic Director, the application form is returned to the strar who creates a student file. The Accounts department receives the application form and rmation from the student file creates an invoice that is sent to the student. Payments are registered on the invoice file. The first batch of student material and tests is issued from only to students who have paid fees (this information is taken from the invoice file).
opic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 2(a).

re the propositions of Putnam's model?
opic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 2(c).

) What do you mean by forking and joining in Activity Diagram? What is a swimlane?
nat are extend and include in use case diagram?
are dependency, aggregation and composition in use case diagram? Explain with ple.
o you mean by an object lifeline and focus of control and which diagram is it required and

Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 12(a), (b) & (c).
Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 12 (b).
Topic: SOFTWARE DESIGN & UML, Long Answer Type Question No. 12 (c).
ic: CODING, TESTING & QUALITY ASSURANCE, Short Answer Type Question No. 8.

short notes on *any three* of the following:
SE tools
ality Assurance
ll Model
tware configuration management
ssion Testing
Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No.

ic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(j).
Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No.

Topic: INTRODUCTION TO SOFTWARE ENGINEERING, Long Answer Type Question No.

ic: CODING, TESTING & QUALITY ASSURANCE, Long Answer Type Question No. 7(k).