

## TYPICAL QUESTIONS & ANSWERS

### PART I

#### OBJECTIVE TYPE QUESTIONS

Each Question carries 2 marks.

Choose the correct or best alternative in the following:

- Q.1** The most important feature of spiral model is  
(A) requirement analysis. (B) risk management.  
(C) quality management. (D) configuration management.

**Ans: B**

- Q.2** The worst type of coupling is  
(A) Data coupling. (B) control coupling.  
(C) stamp coupling. (D) content coupling.

**Ans: D**

- Q.3** IEEE 830-1993 is a IEEE recommended standard for  
(A) Software requirement specification.  
(B) Software design.  
(C) Testing.  
(D) Both (A) and (B)

**Ans: A**

- Q.4** One of the fault base testing techniques is  
(A) unit testing. (B) beta testing.  
(C) Stress testing. (D) mutation testing.

**Ans: D**

- Q.5** Changes made to an information system to add the desired but not necessarily the required features is called  
(A) Preventative maintenance.  
(B) Adaptive maintenance.  
(C) Corrective maintenance.  
(D) Perfective maintenance.

**Ans: D**

- Q.6** All the modules of the system are integrated and tested as complete system in the case of  
(A) Bottom up testing (B) Top-down testing  
(C) Sandwich testing (D) Big-Bang testing

**Ans: D**

- Q.7** If every requirement stated in the Software Requirement Specification (SRS) has only one interpretation, SRS is said to be
- (A) correct. (B) unambiguous.  
(C) consistent. (D) verifiable.

**Ans: B**

- Q.8** A fault simulation testing technique is
- (A) Mutation testing (B) Stress testing  
(C) Black box testing (D) White box testing

**Ans: A**

- Q.9** Modules X and Y operate on the same input and output data, then the cohesion is
- (A) Sequential (B) Communicational  
(C) Procedural (D) Logical

**Ans: B**

- Q.10** If the objects focus on the problem domain, then we are concerned with
- (A) Object Oriented Analysis.  
(B) Object Oriented Design  
(C) Object Oriented Analysis & Design  
(D) None of the above

**Ans: A**

- Q.11** SRS is also known as specification of
- (A) White box testing (B) Stress testing  
(C) Integrated testing (D) Black box testing

**Ans: D**

- Q.12** The model in which the requirements are implemented by category is
- (A) Evolutionary Development Model  
(B) Waterfall Model  
(C) Prototyping  
(D) Iterative Enhancement Model

**Ans: A**

- Q.13** SRD stands for
- (A) Software requirements definition  
(B) Structured requirements definition  
(C) Software requirements diagram  
(D) Structured requirements diagram

**Ans: B**

- Q.14** A COCOMO model is

- (A) Common Cost Estimation Model.
- (B) Constructive Cost Estimation Model.
- (C) Complete Cost Estimation Model.
- (D) Comprehensive Cost Estimation Model.

**Ans: B**

- Q.15** Which of the following statements is true
- (A) Abstract data types are the same as classes
  - (B) Abstract data types do not allow inheritance
  - (C) Classes cannot inherit from the same base class
  - (D) Object have state and behavior

**Ans: B**

- Q.16** The desired level of coupling is
- (A) No coupling
  - (B) Control coupling
  - (C) Common coupling
  - (D) Data coupling

**Ans: D**

- Q.17** In the spiral model 'risk analysis' is performed
- (A) In the first loop
  - (B) in the first and second loop
  - (C) In every loop
  - (D) before using spiral model

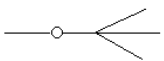
**Ans: C**

- Q.18** For a well understood data processing application it is best to use
- (A) The waterfall model
  - (B) prototyping model
  - (C) the evolutionary model
  - (D) the spiral model

**Ans: A**

- Q.19** Coupling and cohesion can be represented using a
- (A) cause-effect graph
  - (B) dependence matrix
  - (C) Structure chart
  - (D) SRS

**Ans: B**

- Q.20** The symbol  represents
- (A) mandatory 1 cardinality
  - (B) mandatory many cardinality
  - (C) optional 0 or 1 cardinality
  - (D) optional zero-many cardinality

**Ans: D**

- Q.21** Each time a defect gets detected and fixed, the reliability of a software product
- (A) increases.
  - (B) decreases.
  - (C) remains constant.
  - (D) cannot say anything.

**Ans: A**

- Q.22** Output comparators are used in  
(A) static testing of single module  
(B) dynamic testing of single module  
(C) static testing of single and multiple module  
(D) dynamic testing of single and multiple module

**Ans: D**

- Q.23** The feature of the object oriented paradigm which helps code reuse is  
(A) object. (B) class.  
(C) inheritance. (D) aggregation.

**Ans: C**

- Q.24** The level at which the software uses scarce resources is  
(A) reliability (B) efficiency  
(C) portability (D) all of the above

**Ans: B**

- Q.25** If every requirement can be checked by a cost-effective process, then the SRS is  
(A) verifiable (B) traceable  
(C) modifiable (D) complete

**Ans: A**

- Q.26** Modifying the software to match changes in the ever changing environment is called  
(A) adaptive maintenance (B) corrective maintenance  
(C) perfective maintenance (D) preventive maintenance

**Ans: A**

- Q.27** All activities lying on critical path have slack time equal to  
(A) 0 (B) 1  
(C) 2 (D) None of above

**Ans: A**

- Q.28** Alpha and Beta Testing are forms of  
(A) Acceptance testing (B) Integration testing  
(C) System Testing (D) Unit testing

**Ans: A**

- Q.29** An object encapsulates  
(A) Data (B) Behaviour  
(C) State (D) Both Data and behaviour

**Ans: D**

- Q.30** In function point analysis, number of general system characteristics used to rate a system are
- (A) 10 (B) 14  
(C) 20 (D) 12

**Ans: B**

- Q.31** Aggregation represents
- (A) is\_a relationship (B) part\_of relationship  
(C) composed\_of relationship (D) none of above

**Ans: C**

- Q.32** If P is risk probability, L is loss, then Risk Exposure (RE) is computed as
- (A)  $RE = P/L$  (B)  $RE = P + L$   
(C)  $RE = P*L$  (D)  $RE = 2* P *L$

**Ans: C**

- Q.33** Number of clauses used in ISO 9001 to specify quality system requirements are:
- (A) 15 (B) 20  
(C) 25 (D) 28

**Ans: B**

- Q.34** ER model shows the
- (A) Static view. (B) Functional view.  
(C) Dynamic view. (D) All the above.

**Ans: A**

- Q.35** The tools that support different stages of software development life cycle are called:
- (A) CASE Tools (B) CAME tools  
(C) CAQE tools (D) CARE tools

**Ans: A**

- Q.36** Changes made to the system to reduce the future system failure chances is called
- (A) Preventive Maintenance (B) Adaptive Maintenance  
(C) Corrective Maintenance (D) Perfective Maintenance

**Ans: A**

- Q.37** Requirements can be refined using
- (A) The waterfall model (B) prototyping model  
(C) the evolutionary model (D) the spiral model

**Ans: B**

- Q.38** The model that assumes that effort and development time are functions of product size alone is
- (A) Basic COCOMO model                      (B) Intermediate COCOMO model  
(C) Detailed COCOMO model                (D) All the three COCOMO models

**Ans: A**

- Q.39** Structured charts are a product of
- (A) requirements gathering                      (B) requirements analysis  
(C) design    (D) coding

**Ans: C**

- Q.40** The problem that threatens the success of a project but which has not yet happened is a
- (A) bug    (B) error  
(C) risk    (D) failure

**Ans: C**

- Q.41** The main purpose of integration testing is to find
- (A) design errors                                      (B) analysis errors  
(C) procedure errors                                (D) interface errors

**Ans: D**

- Q.42** Pseudocode can replace
- (A) flowcharts    (B) structure charts  
(C) decision tables                                      (D) cause-effect graphs

**Ans: A**

- Q.43** If a program in its functioning has not met user requirements in some way, then it is
- (A) an error.    (B) a failure.  
(C) a fault.    (D) a defect.

**Ans: D**

- Q.44** The testing that focuses on the variables is called
- (A) black box testing                                (B) white box testing  
(C) data variable testing                              (D) data flow testing

**Ans: A**

- Q.45** CASE Tool is
- (A) Computer Aided Software Engineering  
(B) Component Aided Software Engineering  
(C) Constructive Aided Software Engineering  
(D) Computer Analysis Software Engineering

**Ans: A**

- Q.46** Software consists of
- (A) Set of instructions + operating procedures
  - (B) Programs + documentation + operating procedures
  - (C) Programs + hardware manuals
  - (D) Set of programs

**Ans: B**

- Q.47** Which is the most important feature of spiral model?
- (A) Quality management
  - (B) Risk management
  - (C) Performance management
  - (D) Efficiency management

**Ans: B**

- Q.48** Which phase is not available in software life cycle?
- (A) Coding
  - (B) Testing
  - (C) Maintenance
  - (D) Abstraction

**Ans: D**

- Q.49** Which is not a step of requirement engineering?
- (A) Requirements elicitation
  - (B) Requirements analysis
  - (C) Requirements design
  - (D) Requirements documentation

**Ans: C**

- Q.50** FAST stands for
- (A) Functional Application Specification Technique
  - (B) Fast Application Specification Technique
  - (C) Facilitated Application Specification Technique
  - (D) None of the above

**Ans: C**

- Q.51** For a function of two variables, boundary value analysis yields
- (A)  $4n + 3$  test cases
  - (B)  $4n + 1$  test cases
  - (C)  $n + 4$
  - (D) None of the above

**Ans: B**

- Q.52** Site for Alpha Testing is
- (A) Software Company
  - (B) Installation place
  - (C) Any where
  - (D) None of the above

**Ans: A**

- Q.53** Which is not a size metric?
- (A) LOC
  - (B) Function count

- (C) Program length (D) Cyclomatic complexity

**Ans: D**

- Q.54** As the reliability increases, failure intensity  
(A) decreases (B) increases  
(C) no effect (D) none of the above

**Ans: A**

- Q.55** Software deteriorates rather than wears out because  
(A) software suffers from exposure to hostile environments.  
(B) defects are more likely to arise after software has been used often.  
(C) multiple change requests introduce errors in component interactions.  
(D) software spare parts become harder to order.

**Ans: B**

- Q.56** What are the three generic phases of software engineering?  
(A) Definition, development, support  
(B) What, how, where  
(C) Programming, debugging, maintenance  
(D) Analysis, design, testing

**Ans: A**

- Q.57** The spiral model of software development  
(A) Ends with the delivery of the software product  
(B) Is more chaotic than the incremental model  
(C) Includes project risks evaluation during each iteration  
(D) All of the above

**Ans: C**

- Q.58** Which of these terms is a level name in the Capability Maturity Model?  
(A) Ad hoc (B) Repeatable  
(C) Reusable (D) Organized

**Ans: C**

- Q.59** Which of the items listed below is not one of the software engineering layers?  
(A) Process (B) Manufacturing  
(C) Methods (D) Tools

**Ans: B**

- Q.60** Which of the following are advantages of using LOC (lines of code) as a size-oriented metric?  
(A) LOC is easily computed.  
(B) LOC is a language dependent measure.



- (C) LOC is a language independent measure.
- (D) LOC can be computed before a design is completed.

**Ans: A**

- Q.61** Top down approach is used for
- (A) development.
  - (B) identification of faults.
  - (C) testing and validation.
  - (D) reverse engineering.

**Ans: A**

- Q.62** Which of the following is not an attribute of software engineering
- (A) Efficiency.
  - (B) Scalability.
  - (C) Dependability.
  - (D) Usability.

**Ans: C**

- Q.63** A key concept of quality control is that all work products
- (A) are delivered on time and under budget
  - (B) have complete documentation
  - (C) have measurable specification for process outputs
  - (D) are thoroughly tested before delivery to the customer

**Ans: C**

- Q.64** The ISO quality assurance standard that applies to software engineering is
- (A) ISO 9000
  - (B) ISO 9001
  - (C) ISO 9002
  - (D) ISO 9003

**Ans: B**

- Q.65** What types of models are created during software requirements analysis?
- (A) Functional and behavioral
  - (B) Algorithmic and data structure
  - (C) Architectural and structural
  - (D) Usability and reliability

**Ans: A**

- Q.66** What is the normal order of activities in which software testing is organized?
- (A) unit, integration, system, validation
  - (B) system, integration, unit, validation
  - (C) unit, integration, validation, system
  - (D) none of the above

**Ans: A**

- Q.67** Software feasibility is based on which of the following
- (A) business and marketing concerns
  - (B) scope, constraints, market
  - (C) technology, finance, time, resources
  - (D) technical prowess of the developers

**Ans: C**

- Q.68** FP-based estimation techniques require problem decomposition based on  
(A) information domain values      (B) project schedule  
(C) software functions              (D) process activities

**Ans: C**

- Q.69** The software metrics chosen by an organization are driven by the business or technical goals an organization wishes to accomplish.  
(A) True                                  (B) False

**Ans: A**

- Q.70** The goal of quality assurance is to provide management with the data needed to determine which software engineers are producing the most defects.  
(A) True                                  (B) False

**Ans: B**

- Q.71** In the context of requirements analysis, partitioning results in the elaboration of data, function, or behavior.  
(A) True                                  (B) False

**Ans: A**

- Q.72** Units and stubs are not needed for unit testing because the modules are tested independently of one another  
(A) True                                  (B) False

**Ans: A**

## PART II

DESCRIPTIVES

Q 1 Define a software process. How do software myths affect a software process ?

**Ans:**Software process is a Coherent set of activities for specifying, designing, implementing and testing **software** systems.

Software myths propagates misinformation and confusion. Software myths had a no of attributes. For instance, they appeared to be reasonable statements of fact, they had an intuitive feel , and they were often promulgated by experienced practitioners who “know the score” .

Q 2 What is the advantage of using prototype software development model instead of waterfall model? Also explain the effect of defining a prototype on the overall cost of the software project?

**Ans:The waterfall model:** This is the classic SDLC model, with a linear and sequential method that has goals for each development phase. The waterfall model simplifies task scheduling, because there are no iterative or overlapping steps. One drawback of the waterfall is that it does not allow for much revision.

**The prototyping model:** In this model, a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.

Q 3 How does the risk factor affect the spiral model of software development?

**Ans:**Risk Analysis phase is the most important part of "Spiral Model". In this phase all possible (and available) alternatives, which can help in developing a cost effective project are analyzed and strategies are decided to use them. This phase has been added specially in order to identify and resolve all the possible risks in the project development. If risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed with the available data and find out possible solution in order to deal with the potential changes in the requirements.

Q 4 Define software reliability. What is the difference between hardware & software reliability?

**Ans:Software reliability** is the probability that software will provide failure-free operation in a fixed environment for a fixed interval of time. Probability of failure is the probability that the software will fail on the next input selected. **Software reliability** is typically measured per some unit of time, whereas probability of failure is generally time independent. These two measures can be easily related if you know the frequency with which inputs are executed per unit of time. Mean-time-to-failure is the average interval of time between failures; this is also sometimes referred to as Mean-time-before-failure.

Software reliability tends to change continually during test periods. While hardware reliability may change during certain periods such as initial burn in or the end of useful life however it has a much greater tendency then software value.

Hardware faults are not physical faults whereas software faults are design faults that are harder to visualise, classify, detect and correct. In reality, the division between hardware and software reliability is somewhat artificial. Both may be defined in the same way therefore one may combine hardware and software reliability to get system reliability. The source of failure in hardware has generally been physical deterioration.

**Q 5 Explain the following reliability model:  
(i) Basic model (ii) Logarithmic (iii) Poisson Model**

Ans: **(i) Basic Execution Time Model**

The model was developed by J.D. MUSA (MUSA79) in 1979 and is based on execution time. It is assumed that failures may occur according to a non-homogeneous Poisson Process (NHPP). Real world events may be described using Poisson processes. Example of Poisson processes are:

- Number of telephone calls expected in a given period of time
- Expected number of road accidents in a given period of time
- Expected number of persons visiting in a shopping mall in a given period of time.

In this model, the decrease in failure intensity, as a function of the number of failures observed, is constant and is given as:

$$\lambda(\mu) = \lambda_0 \left( 1 - \frac{\mu}{V_0} \right)$$

Where: Initial failure intensity at the start of execution.

$V_0$ : Number of failures experienced, if program is executed for infinite time period.

$\mu$  : Average or expected number of failures experienced at a given point in time.

This model implies a uniform operational profile. If all input classes are selected equally often, the various faults have an equal probability of manifesting themselves. The correction of any of those faults then contributes an equal decrease in the failure intensity. The negative sign shown that there is a negative slope meaning thereby a decrementing trend in failure intensity.

**(ii) & (iii) Logarithmic Poisson Execution Time Model**

This model is also developed by Musa et. Al. (MUSA79). The failure intensity function is different here as compared to Bias model. In this case, failure intensity function (decrement per failure) decreases exponentially whereas it is constant for basic model.

This failure intensity function is given as:

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

Where  $\theta$  is called the failure intensity decay parameter.

The  $\theta$  represents the relative change of failure intensity per failure experienced. Failure intensity of the logarithmic Poisson execution time model drops more rapidly initially than that of the basic model and later it drops more slowly.

The expression for failure intensity is given as:

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

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

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

**Q 6 Distinguish software faults and software failures**

**Ans:** In case of a failure, the software does not do what the user expects. A fault is a programming error that may or may not actually manifest as a failure. A fault can also be described as an error in the correctness of the semantic of a computer program. A fault will become a failure if the exact computation conditions are met, one of them being that the faulty portion of computer software executes on the CPU. A fault can also turn into a failure when the software is ported to a different hardware platform or a different compiler, or when the software gets extended.

**Q 7 Why is SRS also known as the blackbox specification of system ?**

**Ans:** SRS document is a contract between the development team and the customer. Once the SRS document is approved by the customer, any subsequent controversies are settled by referring the SRS document. The SRS document is known as black-box specification. Since the system is considered as a black box whose internal details are not known and only its visible external (i.e. input/output) behaviour is documented. SRS document concentrates on what needs to be done and carefully avoids the solution ("how to do") aspects. The SRS document serves as a contract between development team and the customer. SRS should be carefully written. The requirements at the SRS stage are written using end-user terminology. If necessary later a formal requirement specification may be developed from it.

**Q 8 Write short notes on**

- (i) Configuration Management
- (ii) Key process areas of Capability Maturity model(CMM)

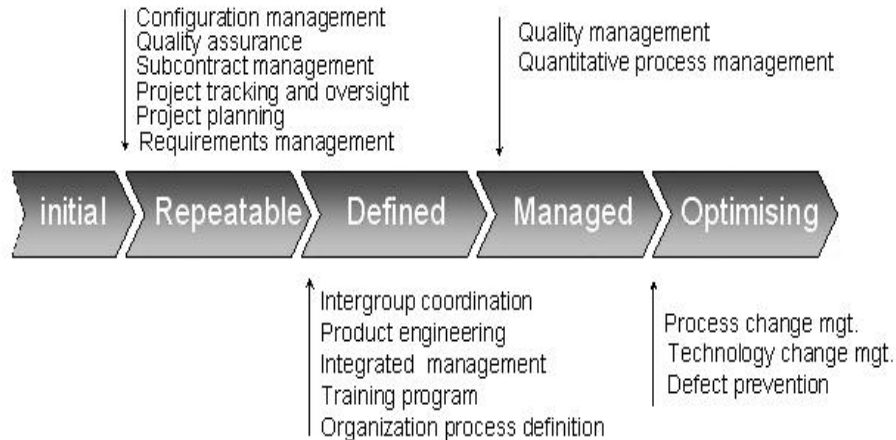
**Ans:** (i) **Software configuration** management is a set of tracking and control activities that begin when a software project begins and terminate only when the software is taken out of operation. **Configuration control** has the following meanings:

- The management of security features and assurances through control of changes made to hardware, software, firmware, documentation, test, test fixtures and test documentation of an automated information system, throughout the development and operational life of a system. *Source Code Management* or *revision control* is part of this.
- The control of changes--including the recording thereof--that are made to the hardware, software, firmware, and documentation throughout the system lifecycle.
- The control and adaption of the evolution of complex systems. It is the discipline of keeping evolving software products under control, and thus contributes to satisfying quality and delay constraints. Software configuration management (or SCM) can be divided into two areas. The first (and older) area of SCM concerns the storage of the entities produced during the software development project, sometimes referred to as component repository management. The second area concerns the activities performed for the production and/or change of these entities; the term engineering support is often used to refer this second area.
- After establishing a configuration, such as that of a telecommunications or computer system, the evaluating and approving changes to the configuration and to the interrelationships among system components.
- In distributed-queue dual-bus (DQDB) networks, the function that ensures the resources of all nodes of a DQDB network are configured into a correct

dual-bus topology. The functions that are managed include the head of bus, external timing source, and default slot generator functions.

**(ii) Key process areas of Capability Maturity model (CMM)**

Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief.



Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process. The key process areas at Level 2 focus on the project's concerns related to establishing basic project management controls. They are Requirements Management, Project Planning, Project Tracking and Oversight, Subcontract Management, Quality Assurance, and Configuration Management.

The key process areas at Level 3 address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective engineering and management processes across all projects. They are Organization Process Focus, Organization Process Definition, Training Program, Integrated Management, Product Engineering, Intergroup Coordination, and Peer Reviews.

The key process areas at Level 4 focus on establishing a quantitative understanding of both the process and the work products being built. They are Quantitative Process Management and Quality Management.

The key process areas at Level 5 cover the issues that both the organization and the projects must address to implement continual, measurable software process improvement. They are Defect Prevention, Technology Change Management, and Process Change Management.

Each key process area is described in terms of the key practices that contribute to satisfying its goals. The key practices describe the infrastructure and activities that contribute most to the effective implementation and institutionalization of the key process area.

**Q 9 Explain cause effect graphing .**

**Ans:** *Cause-effect graphing* is a test case design technique that provides a concise representation of logical conditions and corresponding actions.

There are four steps:

1. *Causes* (input conditions) and *effects* (actions) are listed for a module and an identifier is assigned to each.

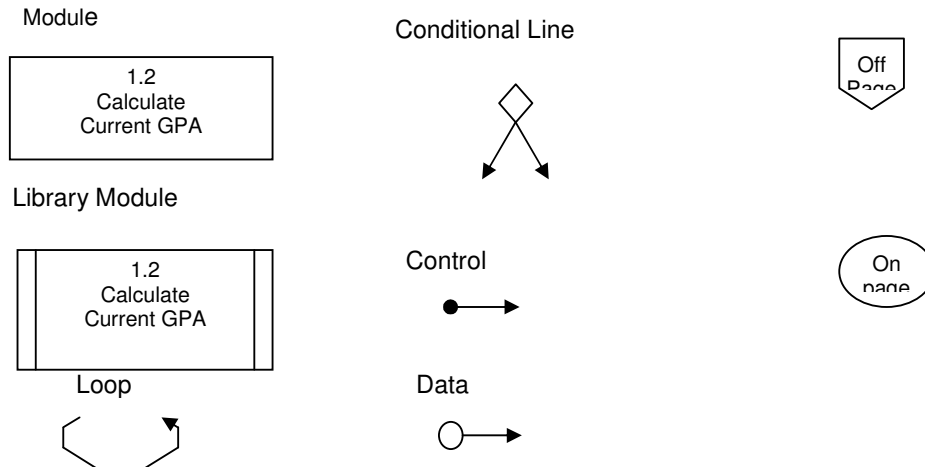
2. A cause-effect graph is developed.
3. The graph is converted to a decision table.
4. Decision table rules are converted to test cases.

**Q10** Write a short note on structure chart.

(6)

**Ans:** **Structure Chart** is an important program design technique and shows all components of code in a hierarchical format.

### Structure Chart Elements

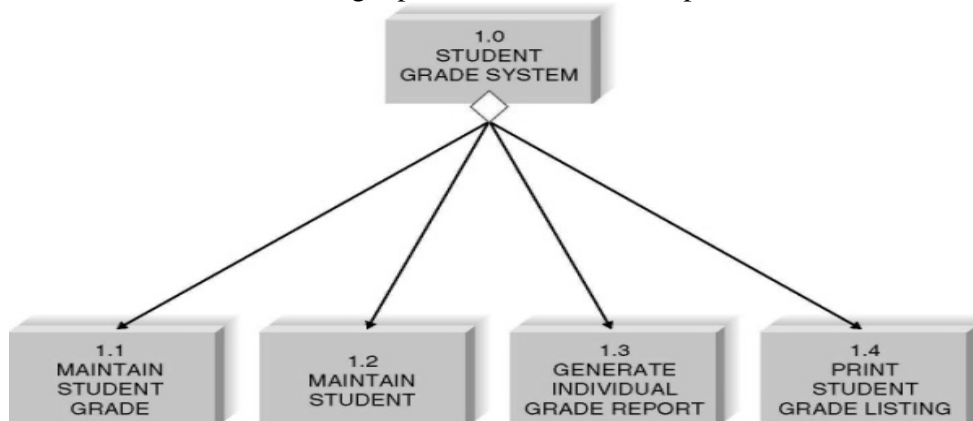


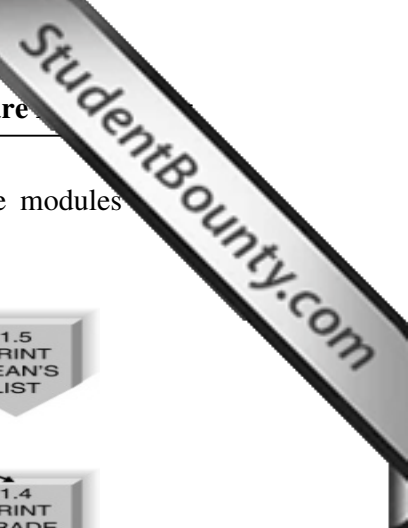
### Building the Structure Chart

- Processes in the DFD tend to represent one module on the structure chart
  - Afferent processes – provide inputs to system
  - Central processes – perform critical system operations
  - Efferent processes – handle system outputs
- The DFD leveling can correspond to the structure chart hierarchy

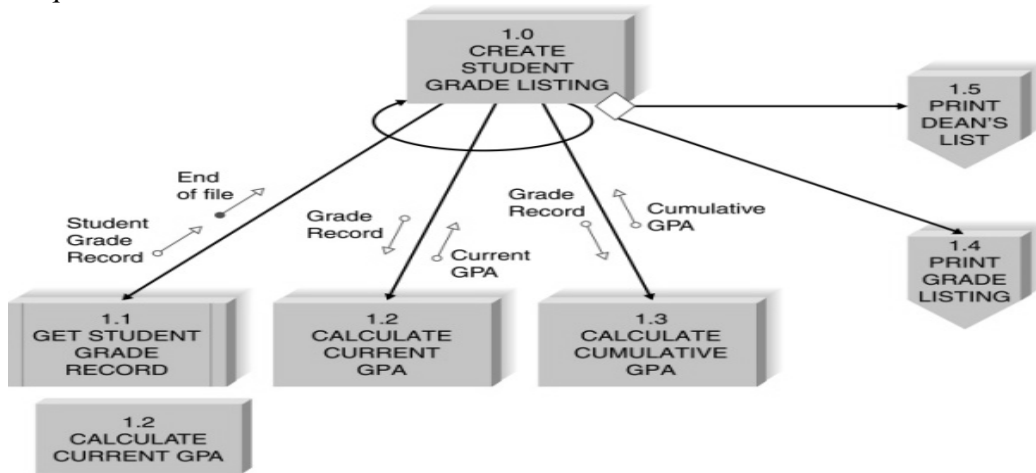
### Types of Structure Charts

- Transaction structure – control module calls subordinate modules, each of which handles a particular transaction
  - Many afferent processes
  - Few efferent processes
  - Higher up levels of structure chart
  - Using inputs to create a new output





- Transform structure – control module calls several subordinate modules in sequence



- Each subordinate performs a step in a process that transforms an input into an output
  - Few afferent processes
  - Many efferent processes
  - Lower levels of structure chart
  - Coordinates the movement of outputs

**Steps in Building the Structure Chart**

1. Identify top level modules and decompose them into lower levels
2. Add control connections
3. Add couples
4. Review and revise again and again until complete

**Design Guidelines**

- High quality structure charts result in programs that are modular, reusable and easy to implement.
- Measures include:
  - Cohesion
  - Coupling
  - Appropriate levels of fan-in and fan-out

**Q 11** For the DFD given in Fig.2 design the structure chart. (8)

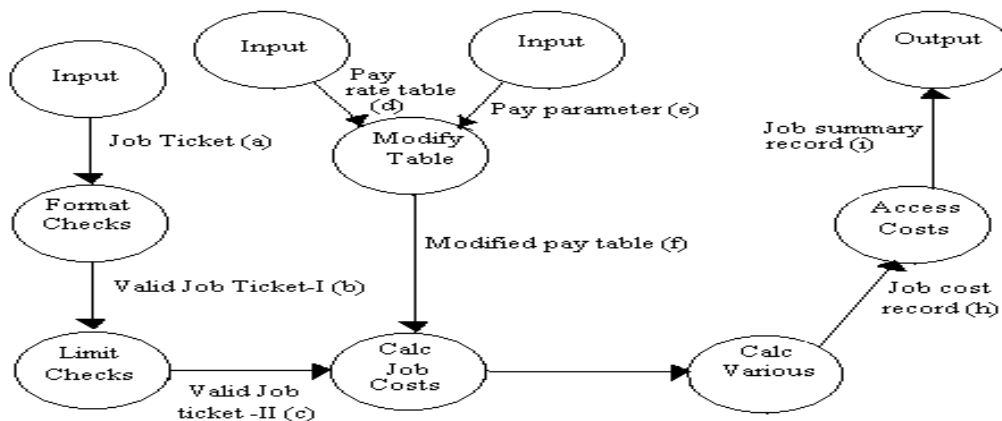
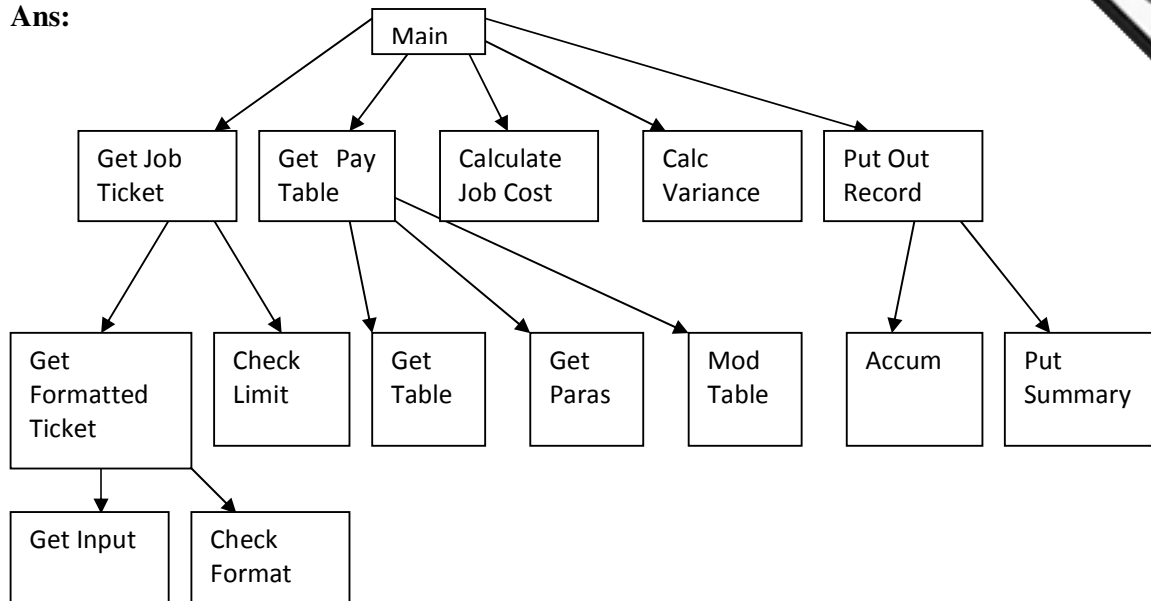


FIG. 2



Ans:



**Q12 Compare the basic COCOMO model with the detailed COCOMO model .**

**Ans.** COCOMO consists of a hierarchy of three increasingly detailed and accurate forms.

- Basic COCOMO - is a static single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code.
- Intermediate COCOMO - computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes.
- Embedded COCOMO - incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

**Q 13 Define software testing. Explain various level of testing .**

**Ans:** Software testing is a process used to help identify the correctness, completeness and quality of developed computer software. With that in mind, testing can never completely establish the correctness of computer software. Only the process of formal verification can prove that there are no defects.

“Testing is the process of demonstrating that defects are not present in the application that was developed.”

“Testing is the activity or process which shows or demonstrates that a program or system performs all intended functions correctly.”

“Testing is the activity of establishing the necessary “confidence” that a program or system does what it is supposed to do, based on the set of requirements that the user has specified.”

Testing is a process of executing a program with the intent of finding an error. various level of testing are :

- Unit Testing
- Integrating testing

Validation testing  
System testing

**Q 14 Differentiate between functional testing and structural testing.**

**Ans:** Functional testing means behavioural testing or **Black box** testing. In this techniques, tester design test case with the behaviour of the modules.  
Structural testing means **White Box** testing. In this testing, tester design test cases from the structure of the module.

**Q 15 Explain some of the limitations of testing.**

**Ans:** Though testing is an important part of system development and leads to a valid, verified and efficient system, it also faces some limitation in its scope.

**Following are some of such limitations.**

- It is very difficult to trace out logical errors through Testing.
- Stress testing or load tests are not the realistic options and hence, it cannot be defined that how application or module will be reacting at heavy data loads.
- In Integration testing, skeletons of different modules are used, which cannot describe the full functioning and in-turn the complete behavior of module they are representing.
- Being performed at later stages, testing may lead to a complete re-development of the module under testing and hence putting all effects in vain.

**Q 16 Why is maintenance of a software important? Discuss some of the problems that are faced during maintenance of software.**

**Ans:** The modification of a software product, after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.

Maintenance is an important part of the software life-cycle. It is expensive in manpower and resources, and one of the aims of software engineering is to reduce its cost.

The most important problem during maintenance is the before correcting or modifying a program, the programmer must first understand it.

**The problems are:**

- Often another person or group of persons working over the years in isolation from each other writes the program.
- Often the program is changed by person who did not understand it clearly, resulting in a deterioration of the program's original organization.
- There is a high staff turnover within information technology industry. Due to this persons who are not the original authors maintain many systems. These persons may not have adequate knowledge about the system.
- Some problems only become clearer when a system is in use. Many users know what they want but lack the ability to express it in a form understandable to programmers. This is primarily due to information gap.

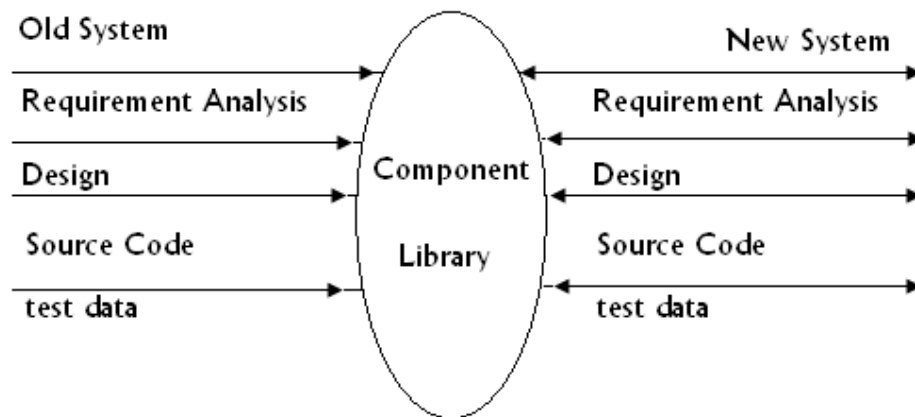
**Q 17 Explain the reuse maintenance model with the help of a diagram.**

**Ans:** Reuse maintenance model: This model is based on the principal that maintenance could be viewed as an activity involving the reuse of existing program components.

**The reuse model has four main steps:**

1. Identification of the parts of the old system that are candidates for reuse.
2. Understanding these system parts.
3. Modification of the Old system parts appropriate to the new requirements.
4. Integration of the modified parts into the new system.

With the five reuse model, the starting point may be any phase of the life cycle. The requirements, the design, the code or the test data :- unlike other models.



**Q 18 Differentiate between object oriented and function oriented design .**

**Ans:**• Function-oriented design relies on identifying functions which transform their inputs to create outputs. In most systems, functions share some global system state.

- The functional design process involves identifying data transformations in the system, decomposing functions into a hierarchy of sub-functions, describing the operation and interface of each system entity and documenting the flow of control in the system.
- Data-flow diagrams are a means of documenting end-to-end data flow through a system. They do not include control information. Structure charts are a way of representing the hierarchical organization of a system. Control may be documented using a program description language (PDL).
- Data-flow diagrams can be implemented directly as a set of cooperating sequential processes. Each transform in the data-flow diagram is implemented as a separate process. Alternatively, they can be realized as a number of procedures in a sequential program.
- Functional design and object-oriented design usually result in totally different system decompositions. However, the most appropriate design strategy is often a heterogeneous one where both functional and object-oriented approaches are used.

**Q 19 How is software design different from coding ?**

**Ans:** Points of difference between software design and coding can be laid down as under:

**Design :**

1. Design is most crucial and time-consuming activity
2. Screen of the system depends on the correct design specifications which is a key activity of the process.
3. Software design is based on the findings collected in the initial investigation phase.
4. Design includes the following:
  - (i) User interface design
  - (ii) Process Design
  - (iii) Database design
5. Designs are transformed into actual code or program during the implementation phase.
6. it is more feasible to rectify design as different users may have conflicting user requirements and only the final and valid design goes for next phase.

**Coding:-**

1. Involves conversion of detailed design specification laid out by designers into actual code, files or database.
2. Less time consuming than the design phase and performed by programmers or coders.
3. More concerned with technical aspect of the software rather than its functional aspect.
4. Different software such as programming languages front-end tools, database management system, utilities etc are used to facilitate the coding process.

**Q 20 What problems arise if two modules have high coupling?**

**Ans:** Coupling means the interconnection of different modules with each other or we can say, it tells about the interrelationship of different modules of a system. A system with high coupling means there are strong interconnections between its modules. If two modules are involved in high coupling, it means their interdependence will be very high. Any changes applied to one module will affect the functionality of the other module. Greater the degree of change, greater will be its effect on the other. As the dependence is higher, such change will affect modules in a negative manner and in-turn, the maintainability of the project is reduced. This will further reduce the reusability factor of individual modules and hence lead to unsophisticated software. So, it is always desirable to have inter-connection & interdependence between modules.

**Q 21 Explain**

- a. the activities of software maintenance. (2)
- b. key process areas of CMM. (6)

**Ans:**

- a) Software maintenance is a broad activity that includes error correction, enhancement of capabilities, deletion of obsolete capabilities and optimization.
- b) Key process areas of CMM are
  1. Requirements management which establishes a common relationship between the customer and developer
  2. Software project planning where reasonable plans for managing the software project are established



**Q 24 What are the three activities of risk assessment? (4)**

**Ans:** The three activities are identifying, analyzing and giving priorities. Risks can be identified by a check list or looking at previously identified risks. Risk analysis involves examining how project outcomes may change with modification of risk input variables. Risk prioritization helps in focus on most severe risks.

**Q 25 What is a modular system? List the important properties of a modular system. (6)**

**Ans:** A modular system consists of well defined manageable units to well defined interfaces among them. Desirable properties are

- Each module is a well defined subsystem useful to others
- Each module has a well defined single purpose
- Modules can be separately compiled and stored in library
- Modules can use other modules
- Modules should be easier to use than build
- Modules should have a simple interface

**Q 26 Define the following:**

- a. **Aggregation among objects.**
- b. **Class.**
- c. **Repeated inheritance.**
- d. **Encapsulation.**
- e. **Scenario.**

(2 x 5)

**Ans:**

a).Aggregation among objects

Aggregation among objects represents a relationship. It is a whole/part relationship. Aggregation may imply containment.

b).Class

A class is a template that specifies the properties of objects. Classes have an interface which consists of the operation, a body which implements the operations and instance variable which contain the state of an object.

c).Repeated inheritance

If a class inherits more than once from the same class then it is referred to as repeated inheritance.

d).Encapsulation

An object encapsulates the data and information it contains and supports a well defined abstraction. Encapsulation leads to the separation of the interface and implementation.

e).Scenario

A scenario is a sequence of events that occur in a particular execution of the system. A scenario can be described textually by enumerating the sequence of events or it can be shown as an event trace diagram in which events between objects are shown.

**Q 27 Explain the following:**

- (i) **Equivalence class testing. (6)**
- (ii) **User and System documentation with examples. (6)**

- (iii) Core dumps. (4)

**Ans:**

- (i) Equivalence class testing is based on partitioning the input domain of a program into a number of equivalence classes so that the test of a representative value of each class is equivalent to testing any other value. Two steps for this method are Identify equivalence class by taking each input condition and partition into valid and invalid classes.  
Generate the test cases using the equivalence class of the previous step. The test cases are generated for the valid and the invalid classes.
- (ii) User documentation contains descriptions of the functions of a system without reference to how these functions are implemented. Examples are installation guide and reference guide. System documents contain all the facets of the system, including analysis, specification design, implementation, testing, security error diagnosis and recovery. Examples are SRS and system test plan.
- (iii) Core dumps are a debugging technique. A printout of all relevant memory locations is obtained and studied. All dumps should be well documented and retained for possible use on subsequent problems. Its advantages are that a complete dump of memory at a crucial time is obtained Require CPU and I/O time and can get expensive if used indiscriminately. Sometimes it is difficult to interpret the dump which is represented using hexadecimal numbers.

- Q 28 Define capability. What are the quantities that determine the degree of capability of a software reliability model? (6)**

**Ans:** Capability refers to the ability of the model to estimate with satisfactory accuracy quantities needed by managers; engineers and users in planning and managing software development or running operational systems. The quantities are MTTF, expected date of reaching reliability, resource and cost requirements to reach the objectives.

- Q.29 Explain the waterfall model. Explain why it is more advantageous than adhoc methods.**

**Ans Waterfall Model:**

1. The **waterfall model** is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, analysis, design (validation), Construction, testing and maintenance.
2. The first formal description of the waterfall model is given by Winston W. Royce in 1950, however, popular in 1970, and further refined by Barry Boehm.
3. To follow the *waterfall model*, one proceeds from one phase to the next in a purely sequential manner. For example, one first completes requirements specifications, which are set in stone. When the requirements are fully completed, one proceeds to design.
4. Process structured as a cascade of phases where output of one is input of next.
5. Many variants of model depending on organization and specific project. However underlying phases are same for all.

**Why Waterfall model is advantageous than Adhoc Methods**

**Ad-hoc Process Models**—“Process capability is unpredictable because the software process is constantly changed or modified as the work progresses. Schedules, budgets, functionality, and product quality are generally inconsistent. Performance depends on the capabilities of individuals and varies with their innate skills, knowledge, and motivations. There are few stable software processes in evidence, and performance can be predicted only by individual rather than organizational capability.” So to overcome this problem waterfall model provides the following advantages:

- Waterfall model is simple to follow, however real projects rarely follows this approach.
- Iteration is not required.
- It is widely use because it is easy.

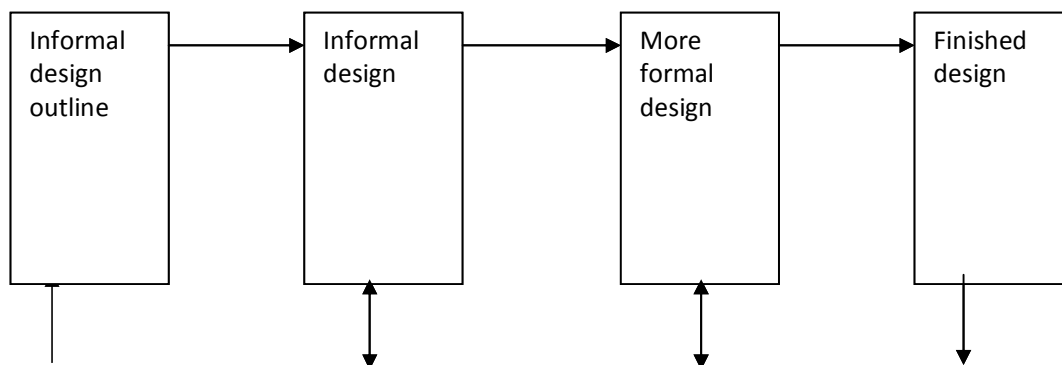
**Q.30 What are the objectives of software design? How do we transform an informal design to a detailed design?**

**Ans Objectives of software design**

The purpose of the design phase is to plan a solution of the problem specified by the requirements document. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed; design takes us toward how to satisfy the needs, so the basic objectives are:

- Identify different types of software, based on the usage.
- Show differences between design and coding.
- Define concepts of structured programming.
- Illustrate some basic design concepts.
- See how to design for testability and maintainability.

Non-formal methods of specification can lead to problems during coding, particularly if the coder is a different person from the designer that is often the case. Software designers do not arrive at a finished design document immediately but develop the design iteratively through a number of different phases. The design process involves adding details as the design is developed with constant backtracking to correct earlier, less formal, designs. The transformation is done as per the following diagram.



**Q.31. Discuss the important issues that a SRS must address.**

**Ans.** SRS is the process of establishing the services, the system should provide of the system in new, it is difficult for software engineer to understand the nature of problem constraints under which it must operates, so software engineer take help of a step called requirement capture and analysis. It is first formal document



produce in the software development process and it serves as a basis for the contract between a producer and a software developer/supplier.

**The important issues that a SRS must address are:**

- (a) System goals and requirements are different: Goal is a more general characteristics." e. g. Whole system should be designed in a user friendly manner or more robust system".  
Requests are more testable in nature. For example all users command selection should be only using pop up menus.
- (b) Request Definition: A statement in a natural language stating what services the system should be expected to provide. It should be understandable by clients, contractors, management & users.
- (c) Request Specification: A structured document maintaining the services in more details than definition, and precise enough to act as a contract. It should be understandable by technical staff both at a developer and producer's place.
- (d) Software Specification (Design Specification): It is an abstract design description of the software which is the basis for design and implementation. There should be a clear relationship between this documents and the software request specification. The reader of this document is software engineer, system analyst and project leaders.
- (e) Requirement Capture and Analysis: It is the process of designing the system request captures through: -
  - (i) Observation of existing system.
  - (ii) Discussion with potential users, producers.
  - (iii) Personal interviews and task analysis.
  - (iv) Standard published documents/reports from the user.
- (f) Feasibility Study:
  - (i) It is estimate made whether the identified user needs can be satisfied using the current technology.
  - (ii) Whether the proposed system is cost effective.
  - (iii) Whether it is possible to develop the system within the budgetary and time constraints.
- (g) Suggestions for preparing an SRS Document:
  - (i) Only specify external system behavior.
  - (ii) Specifying the constraints on implementation.
  - (iii) It should record for thought about the life cycle of the system.
  - (iv) It should characterize acceptable responses to the undesired events.

**Q.32 Explain throw-away prototyping and evolutionary prototyping. Discuss the differences between the two.**

**Ans Throw-Away Prototyping:** Also called **close ended prototyping**. Throwaway or Rapid Prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. Rapid Prototyping involved creating a working model of various parts of the system at a very early stage, after a relatively short investigation. The method used in building it is usually quite informal, the most important factor being the speed with which the model is provided. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. When

this has been achieved, the prototype model is 'thrown away', and the system is formally developed based on the identified requirements.

The most obvious reason for using Throwaway Prototyping is that it can be done quickly. If the users can get quick feedback on their requirements, they may be able to refine them early in the development of the software. Speed is crucial in implementing a throwaway prototype, since with a limited budget of time and money little can be expended on a prototype that will be discarded. Strength of throwaway prototyping is its ability to construct interfaces that the users can test. The user interface is what the user sees as the system, and by seeing it in front of them, it is much easier to grasp how the system will work.

**Evolutionary prototyping:** Evolutionary Prototyping (also known as breadboard prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built. When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt. "Evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood."

Evolutionary Prototypes have an advantage over Throwaway Prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered. In Evolutionary Prototyping, developers can focus themselves to develop parts of the system that they understand instead of working on developing a whole system.

**Q.33 Explain the cost drivers and EAF of the intermediate COCOMO model.**

**Ans.** There are 15 different attributes, called cost drivers attributes that determine the multiplying factors. These factors depend on product, computer, personnel, and technology attributes also know as project attributes. Of the attributes are required software reliability (RELY), product complexity (CPLX), analyst capability (ACAP), application experience (AEXP), use of modern tools (TOOL), and required development schedule (SCHD). Each cost driver has a rating scale, and for each rating, there is multiplying factor is provided. For eg. for the product attributes RELY, the rating scale is very low, low, nominal, high, and very high. The multiplying factor for these ratings is .75, .88, 1.00, 1.15, and 1.40, respectively. So, if the reliability requirement for the project is judged to be low then the multiplying factor is .75, while if it is judged to be very high the factor is 1.40. The attributes and their multiplying factors for different ratings are shown in the table below

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
<b>Product attributes</b>						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
<b>Hardware attributes</b>						

Run-time performance constraints			1.00	1.11	
Memory constraints			1.00	1.06	1.21
Volatility of the virtual machine environment	0.87		1.00	1.15	1.30
Required turnabout time	0.87		1.00	1.07	1.15
<b>Personnel attributes</b>					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	
<b>Project attributes</b>					
Use of software tools	1.24	1.10	1.00	0.91	0.82
Application of software engineering methods	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10

**Q.34 Compare the following**

- (i) **Productivity and difficulty**
- (ii) **Manpower and development time**
- (iii) **Static single variable model and static multivariable model**
- (iv) **Intermediate and Detailed COCOMO model**

**Ans.****(i) Productivity and difficulty**

**Productivity** refers to metrics and measures of output from production processes, per unit of input. Productivity P may be conceived of as a metrics of the technical or engineering efficiency of production. In software project planning, productivity is defined as the number of lines of code developed per person-month

**Difficulty** The ratio  $(K/t_d^2)$ , where K is software development cost and  $t_d$  is peak development time, is called difficulty and denoted by D, which is measured in person/year.

$$D = (K/t_d^2)$$

The relationship shows that a project is more difficult to develop when the manpower demand is high or when the time schedule is short.

Putnam has observed that productivity is proportional to the difficulty

$$P \propto D^\beta$$

**(ii) Manpower and development time**

**Manpower** may refer to labour Manpower, either an abstract term for human labour effort (as opposed to machines, animals etc.) or the number of human productive units available/needed for professional or other tasks, also used when referring to such personnel as a resource (e.g. "a manpower shortage") and development time is the time required to develop a project.

The **Norden/ Reyleigh** equation represents manpower, measured in person per unit time as a function of time. It is usually expressed in person-year/year (PY/YR).

**(iii) Static single variable model and static multivariable model**

**Static single variable model:** Methods using this model use an equation to estimate the desired value such as cost, time, effort etc. They all depend on same variable used as predictor (say, size). An example of the most common equation is

$$C = a L^b$$

Where C is the cost (effort expressed in the unit of manpower, for e.g. persons-months) and L is the size generally given in the line of code (LOC). The constants a and b are derived from the historical data of the organization. Since a and b depends on the local development environment, these models are not transportable to different organizations.

**Static multivariable model:** They depend on several variables representing various aspects of the software development environment, for e.g. methods used, user participation, customer oriented changes, memory constraints, etc. The model provides relationship between delivered line of source code (L in thousand lines) and effort (E in person-months) and is given by the following equation:

$$E = 5.2 L^{0.91}$$

In the same fashion, duration of development (D in months) is given by

$$D = 4.1 L^{0.36}$$

**(iv) Intermediate and Detailed COCOMO model**

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

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

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

**Ratings**

	Very			Very	Extra
Cost Drivers	Low	Low	Nominal	High	High

**Product attributes**

Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65

**Hardware attributes**

Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	

**Personnel attributes**

Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	

**Project attributes**

Use of software tools	1.24	1.10	1.00	0.91	0.82
Application of software engineering methods	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10

The Intermediate Cocomo formula now takes the form:

$$E = a_i (\text{KLOC})^{(b_i)} \cdot \text{EAF}$$

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

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

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

**Detailed COCOMO**

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

**Q.35 List the points of a simplified design process.**

**Ans.** A design process may include a series of steps followed by designers. Depending on the product or service, some of these stages may be irrelevant, ignored in real-world situations in order to save time, reduce cost, or because they may be redundant in the situation. Typical stages of the simplified design process include:

- **Pre-production design**
  - Design brief - an early often the beginning statement of design goals
  - Analysis- analysis of current design goals
  - Research-investigating similar design solutions in the field or related topics
  - Specification - specifying requirements of a design solution for a product (product design specification) or service.
  - Problem solving - conceptualizing and documenting design solutions
  - Presentation - presenting design solutions
- **Design during production**
  - Development - continuation and improvement of a designed solution
  - Testing – on-site testing a designed solution
- **Post-production design feedback for future designs**
  - Implementation - introducing the designed solution into the environment
  - Evaluation and conclusion - summary of process and results, including constructive criticism and suggestions for future improvements
- **Redesign** - any or all stages in the design process repeated (with corrections made) at any time before, during, or after production.

**Q.36 Explain the following with the help of an example**

- (i) **Common coupling**
- (ii) **Communicational cohesion**
- (iii) **Class diagram**
- (iv) **Structure chart**

**Ans.**

**(i) Common coupling:** Common coupling is when two modules share the same global data (e.g. a global variable). Changing the shared resource implies changing all the modules using it. Diagnosing problems in structures with considerable common coupling is time consuming and difficult. However, this does not mean that the use of global data is necessarily "bad". It does not mean that a software designer must be aware of potential consequences of common coupling and take special care to guard against them.

**(ii) Communicational cohesion:** Communicational cohesion is when parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information). In this all of the elements of a component operate on the same input data or produce the same output data. So we can say if a module performs a series of actions related by a sequence of steps to be followed by the product and all actions to be performed on the same data.

**(iii) Class diagram:** A **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes. The UML specifies **two types of scope for members: instance and classifier**. In the case of instance members, the scope is a specific instance. For attributes, it means that its value can vary between instances. For methods, it means that its invocation affects the instance state, in other words, affects the instance attributes. Otherwise, in the classifier member, the scope is the class. For attributes, it means that its value is equal for all instances. For methods, it means that its invocation do not affect the instance state. Classifier members are commonly recognized as "static" in many programming languages.

(iv) **Structure chart:** A structure chart is a top-down modular design tool constructed of squares representing the different modules in the system, and lines that connect them. The lines represent the connection and or ownership between activities and sub activities as they are used in organization charts.

In structured analysis structure charts are used to specify the high-level design, or architecture, of a computer program. As a design tool, they aid the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. A structure chart depicts

- the size and complexity of the system, and
- number of readily identifiable functions and modules within each function and
- Whether each identifiable function is a manageable entity or should be broken down into smaller components.

**Q.37. Explain Equivalence Class Partitioning and Boundary value analysis. Compare the two.**

**Ans. Equivalence Class Partitioning:** It is a technique in black box testing. It is designed to minimize the number of test cases by dividing tests in such a way that the system is expected to act the same way for all tests of each equivalence partition. Test inputs are selected from each class. Every possible input belongs to one and only one equivalence partition." In another words it is a method that can help you derive test cases. You identify classes of input or output conditions. The rule is that each member in the class causes the same kind of behaviour of the system. In other words, the "Equivalence Class Partitioning" method creates sets of inputs or outputs that are handled in the same way by the application. **E.g.:** If you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data. Using equivalence partitioning method above test cases can be divided into three sets of input data called as classes. Each test case is a representative of respective class.

**Boundary Value Analysis:** It's widely recognized that input values at the extreme ends of input domain cause more errors in system. More application **errors occur at the boundaries** of input domain. 'Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in centre of input domain. Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes. **E.g.** if you divided 1 to 1000 input values in valid data equivalence class, then you can select test case values like: 1, 11, 100, 950 etc.

**Q.38 Explain the following terms with reference to software reliability models.**

- (i) **Predictive validity**
- (ii) **Resource usage**
- (iii) **Applicability**

**Ans. (i) Predictive validity:** It is the capability of the model to predict future failure behaviour from present and past failure behaviour. This capability is significant only when failure behaviour is changing. There are two general

ways of viewing predictive validity based on the two equivalent approaches to characterizing failure random process, namely;

1. the number of failure approach and
2. the failure time approach.

We can visually check the predictive validity by plotting the relative error against the normalised test time.

**(ii) Resource usage:** It is linearly proportional to execution time  $\tau$  and mean failures experienced  $\mu$ . Let  $X_r$  be the usage of resource  $r$ . Then

$$X_r = \theta_r \tau + \mu_r \mu$$

Where  $\theta_r$  is the resource usage per CPU hr. It is nonzero for failure identification personnel ( $\theta_i$ ) and computer time ( $\theta_c$ ). The quantity  $\mu_r$  is the resource usage per failure. It is nonzero for failure identification personnel ( $\mu_i$ ), failure correction personnel ( $\mu_f$ ) and computer time ( $\mu_c$ ).

**(iii) Applicability:** It is another important characteristic of a model. The model should be judged by its degree of applicability across software products that vary in size, structure and function. The model should be usable in different development environments, different operational environments, and different life cycle phases. It is desirable that a model should be robust with respect to deviations from its assumptions, errors in the data or parameters it employs, and unusual conditions.

**Q.39 What are the assumptions of the execution-time component model? Compare the execution-time component for basic model and the logarithmic Poisson model.**

**Ans.** The execution-time component is based on the following assumptions:

- (1) Tests represent the environment in which the program will be used.
- (2) All failures are observed.
- (3) Failure intervals are s-independent of each other.
- (4) Hazard rate is a constant that changes at each fault correction.
- (5) The failure rate is proportional to the s-expected value of the number of faults remaining.
- (6) The fault-correction occurrence rate is proportional to the failure-occurrence rate. Both rates are with respect to execution time.

The two models-basic execution time model and logarithmic Poisson execution time model- have failure intensity functions that differ as functions of execution time. However, the difference between them is best defined in terms of slope or decrement per failure experienced. The decrement in the failure intensity function remains constant for basic model while the decrement per failure decreases exponentially as shown in the following graph.

**Q.40. Describe the various types of restructuring techniques. How does restructuring help in maintaining a program?**

**Ans.** Software restructuring modifies source code and / or data an effort to make it amenable to future changes. In general, restructuring does not modify the overall program architecture. It tends to focus on the design details of individual modules and on local data structures define within the module. There are following types of restructuring techniques:



(a) **Code Restructuring:** It is performed to yield a design that produces the same function but with higher quality than the original program. In general, code-restructuring techniques model program logic using Boolean algebra and then apply a series of transformation rules that yield restructured logic. The objective is to take “spaghetti-bowl” code and derive a procedural design that conforms to the structured programming philosophy.

(b) **Data Restructuring:** Before data restructuring begins, a reverse engineering activity called analysis of source code must be conducted. The intent is to extract data items and objects, to get information on data flow, and to understand the existing data structures that have been implemented. This activity is sometimes called data analysis. Once it has been completed, data redesign commences. Another form of redesign, called data rationalization, which ensures that all data naming conventions conform to local standards and that aliases are eliminated as data flow through the system.

**The restructuring helps in maintaining a program in the following ways:**

- (a) Programs have higher quality, better documentation, less complexity, and conformance to modern software engineering practices and standards.
- (b) Frustration among software engineers who must work on the program is reduced, thereby improving productivity and making learning easier.
- (c) Effort requires performing maintenance activities is reduced.
- (d) Software is easier to test and debug.

**Q.41. Describe the various steps of the reuse-oriented model.**

**Ans.** The reuse-oriented model, also called reuse-oriented development (ROD), is a method of software development in which a program is refined by producing a sequence of prototypes called models, each of which is automatically derived from the preceding one according to a sequence of defined rules.

The reuse-oriented model can reduce the overall cost of software development compared with more tedious manual methods. It can also save time because each phase of the process builds on the previous phase that has already been refined. When carefully carried out, ROD can minimize the likelihood of errors or bugs making their way into the final product.

The reuse-oriented model is not always practical in its pure form because a full repertoire of reusable components may not be available. In such instances, some new program components must be designed. If not thoughtfully done, ROD can lead to compromises in perceived requirements, resulting in a product that does not fully meet the needs of its intended users.

**Q.42 What is ripple effect? How does it affect the stability of a program?**

**Ans.** The **ripple effect** is a term used to describe a situation where, like the ever expanding ripples across water when an object is dropped into it, an effect from an initial state can be followed outwards incrementally. Examples can be found in economics where an individual's reduction in spending reduces the incomes of others and their ability to spend. In sociology it can be observed how social interactions can affect situations not directly related to the initial interaction. and in charitable activities where information can be disseminated and passed from community to community to broaden it's impact.

In software, the effect of a modification may not be local to the modification, but may also affect other portions of the program. There is a ripple effect from the location of the modification to the other parts of the programs that are affected by the modification. One aspect of the ripple effect concerns the performance of the program. The primary attribute affecting the ripple effect as a consequence of a program modification is the stability of the program. Program stability is defined as the resistance to the amplification of changes in the program.

**Q.43 List four reasons why it is difficult to improve software process.**

**Ans** It is difficult to improve software process due to following reasons:

1. Lack of knowledge-Many software developers are not aware of best practices of industry. In fact best practices available in literature are not being used widespread in software development.
2. Not enough time-There is always a shortage of time because upper management are always demanding more software of higher quality in minimum possible time. Unrealistic schedule sometimes leave insufficient time to do the essential project work.
3. Wrong motivations-The process improvement initiatives are taken for wrong reasons like sometimes contractor is demanding achievement of CMM or sometimes senior management is directing the organization to achieve CMM without a clear explanations why improvement was needed and its benefits.
4. Insufficient commitments-The software improvement fails due to lack of true commitment. Management sets no expectations from the development community regarding process improvement.

**Q.44 Discuss the various strategies of design. Which design strategy is most popular and practical?**

**Ans** The most commonly used software design strategy involved decomposing the design into functional components with system state information held in a shared data area.

**The design strategies are:**

1. **Functional design** The system is designed from a functional viewpoint, starting from with a high level view and progressively refining this into a more detailed design. The system state is centralised and shared between the functions operating on that state.
2. **Object-oriented design:** The system is viewed as a collection of objects rather than as functions. Object-oriented design is based on the idea of information hiding. In a object-oriented design, the system state is decentralized and each object manages is own state information

**Q.45 Differentiate between structured analysis and structured design.**

**Ans.** The aim of structured analysis is to transform the textual description of a problem into a graphic model. During structured analysis, functional decomposition of the system takes place, i.e. each function that the system performs is analyzed and hierarchically decomposed into more detailed functions. On the other hand, during structured design all functions identified during structured analysis are mapped to a module structure. This module structures is also

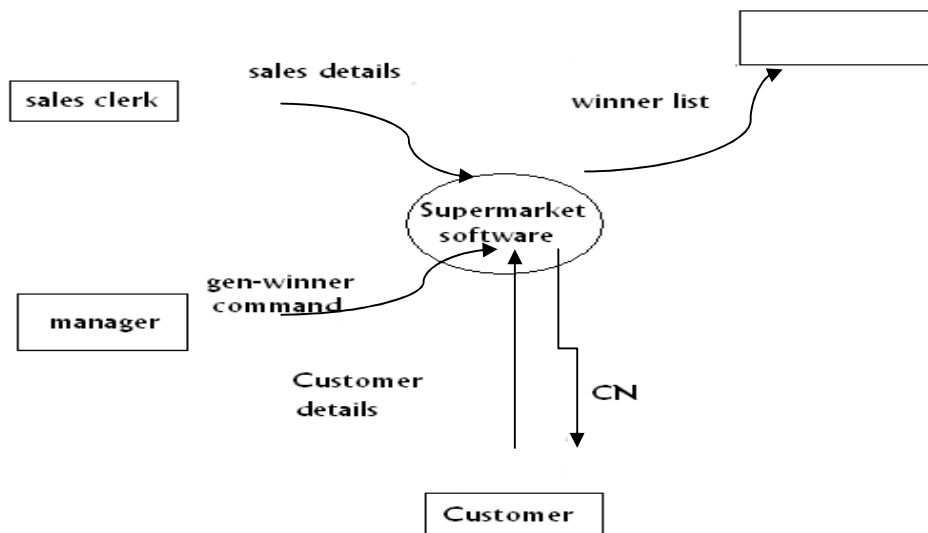
called software architecture for the given problem and it can be directly implemented using a conventional programming language.

The purpose of structured analysis is to capture the detailed structure of the system as perceived by the user, whereas the purpose of structured design is to define the structure of the solution that is suitable for implementation in some programming language.

**Q.46** A supermarket needs to develop software to encourage regular customers. For this, the customer needs to supply his name, address, telephone number and driving license number. A customer is assigned a unique customer number (CN) by the computer. When a customer makes a purchase, the value of the purchase is credited against his CN. At the end of each year, surprise gifts to 10 customers who have made the highest purchase is given. In addition, a 22 carat gold coin is given to every customer who has made a purchase over Rs.10,000/-. The entries are reset on the last day of the year.

- (i) Draw the context diagram
- (ii) Give data dictionary entries for
  - address
  - CN
  - gold-coin-winner-list
  - totalsales

Ans.



Data dictionary entries for:

**address:** name + house# + street# + city + pin

name: string

house#: string

street#: string

city: string

pin: integer

**CN:** {integer|character|spl. character}\*

**Gold-coin-winner-list:** {address}\*

**Total sales:** integer

- Q.47 Explain the following:**
- (i) **Information flow index.**
  - (ii) **Span and average span size for a program.**
  - (iv) **Function points.**

**Ans (i) Information Flow index** Information flow metrics are applied to the components of a system design. For any component 'A', We can define three measures:

1: 'FAN IN' is simply a count of the number of other components that can all, or pass control, to component A.

2: 'FAN OUT' is the number of components that are called by component A.

3: using the following formula derives this. We will call this measure the INFORMATION FLOW index of component A , abbreviated as IF(A).

$$IF(A) = \{FAN\ IN(A) * FAN\ OUT(A)\}^2$$

**(ii) Span and average span size of a program**

Span size indicates the number of statements that pass between successive uses of a variable. For example in the following sample of code, variable 'x' has 4 spans of 10,12,7, and 6 statements while 'y' has 2 spans of 23 and 14 statements. This shows x is being used more than y.

```

...
21 scanf(x,y);
...
32 a=x;
...
45 b=x-y;
...
53 c=x;
...
60 printf(x v);

```

Average span size =  $(\sum \text{span size of a variable in a program}) / \text{no. of span}$

For example

Average span size of variable x in the above sample =  $(10+12+7+6)/4=8.75$

Average span size of variable y =  $(23+14)/2=18.5$

**(iii) Function points:** Function point measures the functionality from the user point of view, that is, on the basis of what the user request and receives in return. Therefore, it deals with the functionality being delivered, and not with the lines of code, source modules, files etc. Measuring size in this way has the advantage that size measure is independent of the technology used to deliver the function.

**Importance of function point:**

- This is independent of the languages tools, or methodology used for implementation.
- They can be estimated from requirement specification or design specification.
- They are directly linked to the statement of request.

- Q.48 Explain the development phases of the detailed COCOMO model.**

**Ans** A software development is carried out in four successive phases, which are as follows:

**1: plan/requirements:** this is the first phase of the development cycle. The requirement is analyzed, the product plan is set up and a full product specification is generated. This phase consumes from 6% to 8% of the effort and 10% to 40% of the development time.

**2: product design:** The second phase of the COCOMO development cycle is concerned with the determination of the product architecture and the specification of the subsystems. This requires 16% to 18% of the normal effort and 19% to 38% of the development time.

**3: programming:** This is the third phase and is divided into sub phases : detailed phase and code/unit phase. This requires 48% to 68% of the effort and 24% to 64% of the development time.

**4: integration/test :** This phase of COCOMO occurs before delivery. This mainly consists of putting the tested parts together and then testing the final product. This requires 16% to 34% of normal effort and 18% to 34% of the development time.

**Q.49 Why does the software design improve when we use object-oriented concepts?**

**Ans.** The software design improves when we use object-oriented concepts because

- 1.Object-orientation works at higher level of abstraction. The development can proceed at the object level and ignore the rest of the system for as long as necessary.
- 2.The data on which a system is based tends to be more stable than the functionality it supports.
- 3.Object-orientated programming encourages and supports good programming techniques.
- 4.Object-oriented programming design and programming promote code re-use.

**Q.50 Describe system testing.**

**Ans. System testing:**

System test are designed to validate a fully developed systems with a view to assuring that it meets its requirements. There are three types of system testing:

**Alpha testing:**

This refers to system testing that is carried out by the test team within the organisation.

- **Beta testing:**

This is the system testing performed by a select group of friendly customers.

- **Acceptance testing:**

this is performed by the customer to determine whether or not to accept the delivery of the system.

**Q.51 Define graph matrix and connection matrix.**

**Ans: Graph matrix:** A graph matrix is a square matrix whose size(i.e., number of rows and columns) is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node, and matrix entries correspond to connections (an edge) between nodes.

**Connection matrix:** In connection matrix, each letter has been replaced with a indicating that a connection exists (zeros have been excluded for clarity).

**Q.52 What are dynamic testing tools? Explain the functions that they must support.**

**Ans. Dynamic testing tools:**

1: **coverage analyzers (execution verifiers):** A coverage analyzer is the most common and important tool for testing. It is often relatively simple. One of the common strategies for testing involves declaring the minimum level of coverage, ordinarily expressed as a percentage of the elemental segments that are exercised in aggregate during the testing process.

2: **output comparators:** these are used in dynamic testing-both single-module and multiple-module(system level) varieties to check that predicted and actual outputs are equivalent. This is also done during regression testing.

3: **Test data generators:**this is a difficult one, and at lease for the present is one for which no general solution exists. one of the practical difficulties with test data generation of sets of inequalities that represent the condition along a chosen path.

4: **Test file generators:** this creates a file of information that is used as the program and does so based on comments given by the user and/or from the data descriptions program's data definition section.

5: **test harness systems:**this is one that is bound around the test object and that permits the easy modification and control of test inputs and output's and provides for online measurement of CI coverage values.

6: **Test archiving systems:** the goal is to keep track of series of tests ant to act as the basis for documenting that the tests have been done and that no defects were found during the process.

**Functions that dynamic testing tools supports:**

1: input setting: selecting of the test data that the test object reads when called.

2: stub processing: handling outputs and selecting inputs when a stub is called.

3: results displays: providing the tester with the values that the test object produces so that they can be validated.

4: test coverage measurement: determining the test effectiveness in terms of the structure of the program.

5: test planning: helping the tester to plan tests so they are both efficient and also effective at forcing discovery of defects.

**Q.53 What is reverse engineering?**

**Ans.** It is a process of analysing software with a view to understanding its design and specification.

- In reverse engineering, source code and executable code are the input.
- It may be part of a re-engineering process but may also be used to re-specify a system for re-implementation.
- Reverse engineering often proceeds re-engineering but is sometimes worth wise in its own right.
- Builds a program database and generates information from this.

- Program understanding tools (browsers, cross reference etc.) may also be used in this process.
- Design and specification may be reverse engineer to :
  - Serves as input to SRS for program replacement.
  - Be available to help program maintenance.

**Q.54 What is structured programming and why is it important?**

**Ans.** Structured programming is a term which was coined in the late 1960's to mean programming without using go to statements, programming using only while loops and if statements as control constructs and designing using a top-down approach. The adoption of structured programming was an important milestone in the development of software engineering because it was the first away from an undisciplined approach to software development. Structured programming means programs can be read sequentially and are therefore easier to understand and inspect.

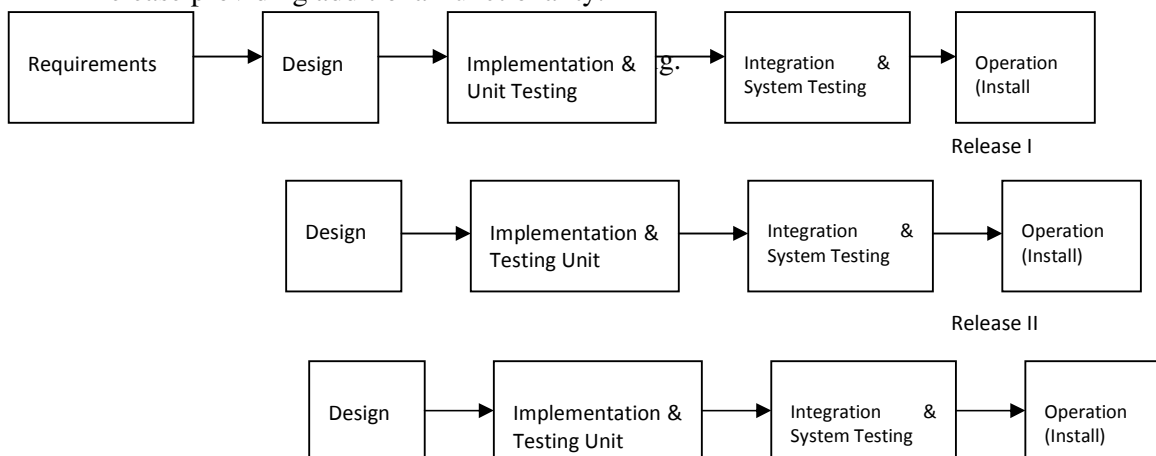
**Q.55 Discuss any two software characteristics.** (5)

**Ans. Software is not manufactured:** The life of a software is from concept exploration to the retirement of the software product. It is one time development effort and continuous maintenance effort in order to keep it operational. However, making 1000 copies of software is not an issue and it does not involve any cost. In case of hardware product, every product costs us due to raw material and other processing expenses. We do not have assembly line in software development. Hence it is not manufactured in the classical sense.

**Software is flexible:** We all feel that software is flexible. A program can be developed to do almost anything. Sometimes this characteristic may be the best and may help us to accommodate any kind of change.

**Q.56 Differentiate between iterative Enhancement Model and Evolutionary Development model.** (5)

**Ans.** Iterative Enhancement Model: This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but these may be conducted in several cycles. A useable product is released at the end of the each cycle, with each release providing additional functionality.



Evolutionary Development Model: Evolutionary development model resembles iterative enhancement model. The same phases as defined for the waterfall model occur here in a cyclical fashion. This model differs from iterative enhancement model in the sense that this does not require a useable product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

(6)

**Q.57 Explain the software life cycle model that incorporates risk factor.**

**Ans.** The problem with traditional software process models is that they do not deal sufficiently with the uncertainty, which is inherent to software projects. Important software projects failed because project risks were neglected and nobody was prepared when something unforeseen happened. Barry Boehm recognized this and tried to incorporate the “project risk” factor into a life cycle model. The result is the spiral model, which was presented in 1986 BOEH86. Each loop of the spiral from X-axis clockwise through 360 degrees represents one phase. One phase is split roughly into four sectors of major activities.

- Planning : Determination of objectives, alternatives and constraints
- Risk Analysis : Analyze alternatives and attempts to identify and resolve the risks involved
- Development : Product development and testing product
- Assessment : Customer evaluation

During the first phase, planning is performed, risks are analyzed, prototypes are built, and customers evaluate the prototype. During the second phase, a more refined prototype is built, requirements are documented and validated, and customers are involved in assessing the new prototype. By the time the third phase begins, risks are known, and a somewhat more traditional development approach. The advantage of this model is the wide range of options to accommodate the good features of other life cycle models. It becomes equivalent to another life cycle model in appropriate situations. It also incorporates software quality objectives into software development. The risk analysis and validation steps eliminate errors in the early phases of development.

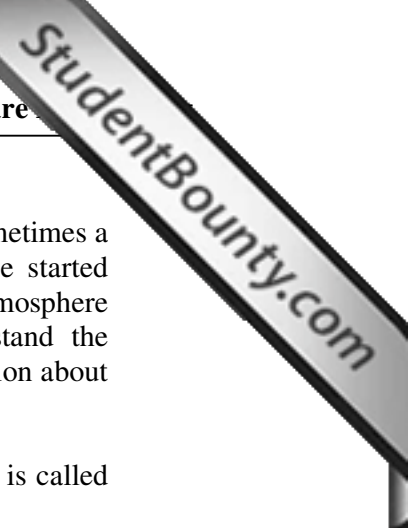
**Q.58 Explain any two requirement elicitation methods.**

**Ans. Interviews:**

After receiving the problem statement from the customer, the first step is to arrange a meeting with the customer. During the meeting or interview, both the parties would like to understand each other. Normally specialized developers often called ‘requirement engineers’ interact with the customer. The objective of conducting an interview is to understand the customer’s expectation from the software. Both parties have different feelings, goals, opinions, vocabularies, understanding, but one thing in common, both want the project to be a success. With this in mind, requirement engineers normally arrange interviews. Requirement engineers must be open minded and should not approach the interview with pre-conceived notions about what is required.

Interview may be open-ended or structured. In an open-ended interview there is no pre-set agenda. Context-free questions may be asked to understand the problem.





and to have an overview of the situation.

In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview. Interview, may be started with simple questions to set people at ease. After making atmosphere comfortable and calm, specific questions may be asked to understand the requirements, The customer may be allowed to voice his or her perception about a possible solution.

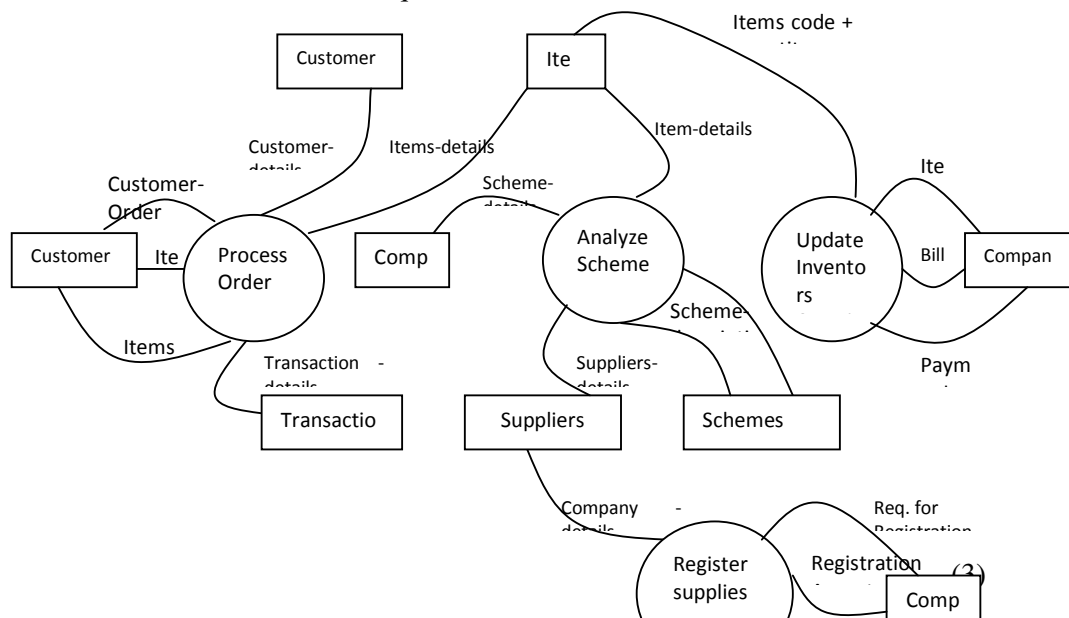
**Facilitated Application Specification Technique**

A team oriented approach is developed for requirements gathering and is called facilitated application specification Techniques (FAST)

This approach encourages the creation of a joint team of customers and developers who work together to understand the expectations and propose of set of requirements. The basic guidelines of FAST are given below :

- Arrange a meeting at a neutral site for developers and customers.
- Establishment of rules for preparation and participation.
- Prepare and informal agenda that encourages free flow of ideas.
- Appoint a facilitator to control the meeting. A facilitator may be a developer, a customer, or an outside expert. Prepare a definition mechanism-Board, flip chart, worksheets, wall stickiest, etc.
- Participants should not criticize or debate.

Q.59 A store is in the business of selling paints and hardware items. A number of reputed companies supply items to the store. New suppliers can also register with the store after providing necessary details. The customer can place the order with the shop telephonically. Or personally. In case items are not available customers are informed. The detail of every new customer is stored in the company’s database for future reference. Regular customers are offered discounts. Additionally details of daily transactions are also maintained. The suppliers from time to time also come up with attractive schemes for the dealers. In case, scheme is attractive for a particular item, the store places order with the company. Details of past schemes are also maintained by the store. The details of each item i.e. item code, quantity available etc. is also maintained. Draw a level 1 DFD for the above requirement.



Q.60 **List any three characteristics of a good SRS.**

**Ans: The SRS should be: Correct, Unambiguous, Complete**

1. Correct: An SRS is correct iff every requirement stated therein is one that the software shall meet.
2. Unambiguous: An SRS is unambiguous iff every requirement stated therein has only one interpretation. Each sentence in SRS should have the unique interpretation.
3. Complete: An SRS is complete iff it includes all significant requirements full labels and references to all figures and diagram and definition of all terms and units of measures

Q.61 **Define the following terms:**

- (i) **Product metrics**
- (ii) **Live variables**
- (iii) **FAN IN** (10)

**Ans:** (i) Product metrics: describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.

(ii) A variable is live from its first to its last reference within a procedure.

(iii) FAN IN is simply a count of the number of other Components that can call, or pass control, to Component A.

Q.62 **Compare the Organic Semi detached and Embedded cocomo modes** (8)

**Ans:**

Mode	Project size	Nature of Project	Innovation	Deadline of the project	Development Environment
Organic	Typical 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typical 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar projects. For example:	Medium	Medium	Medium

		Utility systems like compilers, database systems, editors etc.			
Embedded	Typical over 300 KLOC	Large project, Real time systems, Complex interfaces, very little previous experience. For Example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/customer interfaces required

Q.63 **Why is good design important for a product?** (3)

**Ans:** A good design is the key to successful product. Almost 2000 years ago a Roman Architect recorded the following attributes of a good design:

- Durability
- Utility and
- Charm

A well-designed system is easy to implement, understandable and reliable and allows for smooth evolution.

Q.64 **Define coupling. Discuss various types of coupling.** (8)

**Ans:** Coupling is the measure of the degree of interdependence between modules.

**Type of coupling :** Different types of coupling are content, common, external, control, stamp and data. The strength of coupling from lowest coupling (best) to highest coupling (worst) is given in the figure.

Data coupling	Best
Stamp coupling	↑
Control coupling	
External coupling	
Common coupling	
Content coupling	

Given two procedures A and B, we can identify a number of ways in which they can be coupled.

#### **Data coupling**

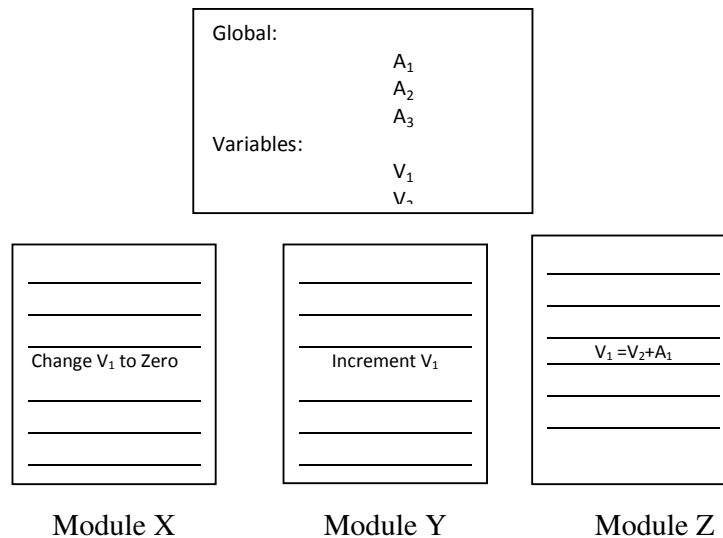
The dependency between module A and B is said to be data coupled if their dependency is based on the fact they communicate by only passing of data. Other than communicating through data, the two modules are independent. A good strategy is to ensure that no module communication contains “tramp data” only the necessary data is passed. Students name, address, course are example of tramp data that are unnecessarily communicated between modules. By ensuring that modules communicate only necessary data, module dependency is minimized.

**Stamp coupling:** Stamp coupling occurs between module A and B when complete data structure is passed from one module to another. Since not all data making up the structure is usually necessary in communication between the modules, stamp coupling typically involves data. If one procedure only needs a part of a data structure, calling modules pass just that part, not the complete data structure.

**Control coupling:** Module A and B are said to be control coupled if they communicate by passing of control information. This is usually accomplished by means of flags that are set by one module and reacted upon by the dependent module.

**External coupling:** A form of coupling in which a module has a dependency to other module, external to the software being developed or to a particular type of hardware. This is basically related to the communication to external tools and devices.

**Common coupling:** With common coupling, module A and module B have shared data. Global data areas are commonly found in programming languages. Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of change. With common coupling, it can be difficult to determine which module is responsible for having set a variable to a particular value. Fig. below shows how common coupling works



**Content coupling:** Content coupling occurs when module A changes data of module B or when control is passed.

Q.65

**Explain the concept of bottom-up, top-down and hybrid design.**

(8)

**Ans:** Bottom up design: This approach leads to a style of design where we decide how to combine these modules to provide larger ones; to combine those to provide even larger ones, and so on, till we arrive at one big module which is the whole of the desired program.

Since the design progressed from bottom layer upwards, the method is called bottom up design. The main argument for this design is that if we start coding a module soon after its design, the chances of recoding is high; but the coded module can be tested and design can be validated sooner than

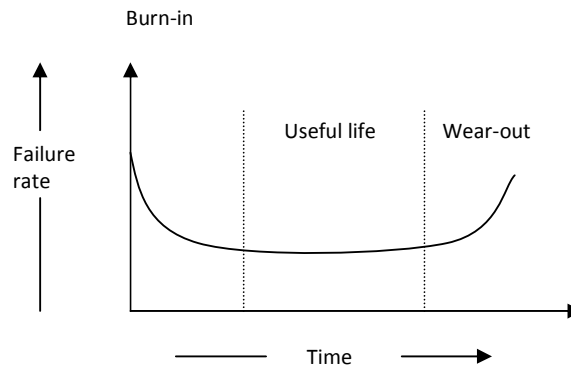
a module whose sub modules have not yet been designed.

**Top down design:** A top down design approach starts by identifying the major modules of the system, decomposing them into their lower level modules and iterating until the desired level of details is achieved. This is stepwise refinement; starting from an abstract design, in each step the design is refined to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly.

**Hybrid design:** Pure top-down or pure bottom up approaches are often not practical therefore hybrid approach which combines the above two approaches is often used.

Q.66 **Explain the bath tub curve of hardware reliability.** (5)

**Ans:** As indicated in the figure below, there are three phase in the life of any hardware component i.e. burn-in, useful life & wear out. In burn-in phase, failure rate is quite high initially, and it starts decreasing gradually as the time progresses. It may be due to initial testing in the premises of the organization. During useful life period, failure rate is approximately constant. Failure rate increases in wear-out phase due to wearing out/aging of components. The best period is useful life period. The shape of this curve is like a “bath tub” and that is why it is known as bath tub curve.



Bath tub curve of hardware reliability

Q.67 **Consider a program for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a, b, c) and values in the interval [0,100]. The program output may be one of the following:**  
**[Not a quadratic equation; Real roots; Imaginary roots; Equal roots].**  
**Design the boundary value test cases and robust test cases for the program.** (16)

**Ans.** Quadratic equation will be of type:

$$ax^2 + bx + c = 0$$

Roots are real if  $(b^2 - 4ac) > 0$

Roots are imaginary if  $(b^2 - 4ac) < 0$

Roots are equal if  $(b^2 - 4ac) = 0$

Equation is not quadratic if  $a=0$

The boundary value test cases are:

Test Case	a	b	c	Expected output
1	0	50	50	Not Quadratic
2	1	50	50	Real Roots
3	50	50	50	Imaginary Roots
4	99	50	50	Imaginary Roots
5	100	50	50	Imaginary Roots
6	50	0	50	Imaginary Roots
7	50	1	50	Imaginary Roots
8	50	99	50	Imaginary Roots
9	50	100	50	Equal Roots
10	50	50	0	Real Roots
11	50	50	1	Real Roots
12	50	50	99	Imaginary Roots
13	50	50	100	Imaginary Roots

As we know, robust test cases are  $6n+1$ . Hence, in 3 variable input cases total number of test cases are 19 as given below:

Test Case	A	B	C	Expected Output
1	-1	50	50	Invalid Input
2	0	50	50	Not Quadratic Equation
3	1	50	50	Real Roots
4	50	50	50	Imaginary Roots
5	99	50	50	Imaginary Roots
6	100	50	50	Imaginary Roots
7	101	50	50	Invalid Input
8	50	-1	50	Invalid Input
9	50	0	50	Imaginary Roots
10	50	1	50	Imaginary Roots
11	50	99	50	Imaginary Roots
12	50	100	50	Equal Roots
13	50	101	50	Invalid Input
14	50	50	-1	Invalid Input
15	50	50	0	Real Roots
16	50	50	1	Real Roots
17	50	50	99	Imaginary Roots
18	50	50	100	Imaginary Roots
19	50	50	101	Invalid Input

**Q.68** Discuss the problems faced during software maintenance (7)

**Ans. Problems During Maintenance** The most important problem during maintenance is that before correcting or modifying a program, the programmer must first understand it. Then, the programmer must understand the impact of the intended change. Few problems are discussed

below.

Often the program is written by another person or group of persons working over the years in isolation from each other.

Often the program is changed by person who did not understand it clearly, resulting in a deterioration of the program's original organization.

Program listing, even those that are well organized, are not structured to support reading or comprehension. We normally read article or book straight through, but with listing, programmer rummages back and forth.

There is a high staff turnover within information Technology industry. Due to this many systems are maintained by persons who are not the original authors. These persons may not have adequate knowledge about the system. This may mean that these persons may introduce changes to programs without being aware of their effects on other parts of the system -the ripple effect. This problem may be worsened by the absence of documentation. Even where it exists, it may be out of date or inadequate.

Some problems only become clearer when a system is in use. Many users know that they want but lack the ability to express it in a form understandable to programmers/analysts. This is primarily due to information gap.

Systems are not designed for change. If there is hardly any scope for change, maintenance will be very difficult. Therefore approach of development should be the production of maintainable software.

**Q.69 Explain acceptance testing and beta testing. (5)**

**Ans. Acceptance Testing and Beta testing:** System tests are designed to validate a fully developed system to assure that it meets its requirements.

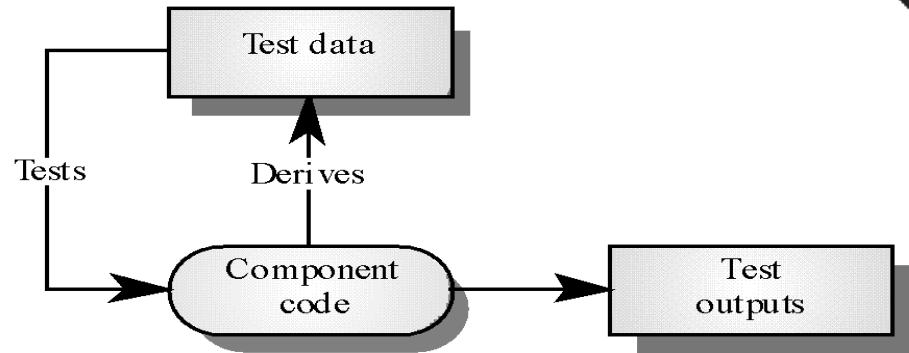
Acceptance and beta testing are form of system testing:

- **Beta testing.** Beta testing is the system testing performed by a select group of friendly customers.
- **Acceptance Testing.** Acceptance testing is the system testing performed by the customer to determine whether he should accept the delivery of the system.

**Q.70. Define the following (9)**

- (i) **Structural testing**
- (ii) **Special value testing**
- (iii) **Mutation testing**

**Ans(i) Structural Testing:** Sometime called white-box testing. Here derivations of test cases are according to the program structure. Knowledge of the program is used to identify additional test cases. The objective of structural testing is to exercise all program statements (not all path combinations)



**(ii) Special Value Testing:** It is the form of functional testing. Special value testing occurs when a tester uses his or her domain knowledge, experience with similar program and information about “soft spots” to dense test case. No guidelines are used other than to use “best engineering judgment”. As a result special value testing is heavily dependent on the abilities of the testing persons.

**(iii) Mutation testing** In mutation testing, the software is first tested by using an initial test suite built up from the different white box testing strategies. After the initial testing is complete, mutation testing is taken up. The idea behind mutation testing is to make few arbitrary changes to a program at a time. Each time the program is changed, it is called as a mutated program and the change effected is called as a mutant. A mutated program is tested against the full test suite of the program. If there exists at least one test case in the test suite for which a mutant gives an incorrect result, then the mutant is said to be dead. If a mutant remains alive even after all the test cases have been exhausted, the test data is enhanced to kill the mutant. A major disadvantage of the mutation-based testing approach is that it is computationally very expensive, since a large number of possible mutants can be generated.

**Q.71. What is CMM? Describe its levels & compare it with ISO 9001?**

**Ans. CAPABILITY MATURITY MODEL (CMM):** CMM is a strategy for improving the software process, irrespective of the actual life cycle model used. Software Engineering Institute (SEI) of Carnegie-Mellon University developed CMM in 1986. CMM is used to judge the maturity of the software processes of an organization and to identify the key practices that are required to increase the maturity of these processes.

**Different levels of CMM:**

1) **Initial (maturity level 1)**-At this, the lowest level, there are essentially no sound software engineering management practices in place in the organization. Instead everything is done on adhoc basis. In maturity level 1 organization, the software process is unpredictable.

2) **Repeatable (maturity level 2)**-At this level, policies for managing a software project and procedures to implement those policies are established. Planning and managing new projects is based on experience with similar projects. An objective in achieving level 2 is to institutionalize effective management processes for software projects, which allow organizations to repeat successful practices, developed on other projects. The software process capability at level 4 organizations can be summarized as disciplined.

3) **Defined (maturity level 3)**- At this level, the standard process for developing and maintaining software across the organization is documented, including both software



engineering and management processes. Processes established at level 3 are used to help the software managers and technical staff to perform more effectively. The software process capability at level 4 organizations can be summarized as standard and constituent.

4) **Managed (maturity level 4)**-At this level, the organization sets quantitative quality goals for both software products and processes. Productivity and quality are measured for important software process activities across all projects as part of an organizational measurement program. The software process capability at level 4 organizations can be summarized as “predictable”.

5) **Optimizing (maturity level 5)**-At this level, the entire organization is focused on continuous process improvement. The organizations have the means to identify weaknesses and strengthen the process proactively, with the goal of preventing the occurrence of defects. . The software process capability at level 4 organizations can be summarized as continuously improving.

#### **ISO 9001 and CMM:**

1: **Management responsibility:** ISO 9001 requires that the quality policy be defined, documented, understood, implemented, and maintained. Whereas in CMM, management responsibility for quality policy and verification activities is primarily addressed in software quality assurance.

2: **Document control:** ISO 9001 requires that the distribution and modification of documents be controlled. In the CMM, the configuration management practices characterizing document control are described in software configuration management.

3: **Purchasing:** ISO 9001 requires that purchased products conform to their specified requirements.

In the CMM, this is addressed in software subcontract management.

4: **Training:** ISO 9001 requires that training needs to be identified and that training be provided. Records of training are maintained.

In CMM, specified trainings are identified in the training and orientation practices in the ability to perform common feature.

5: **Internal quality audit:**ISO 9001 requires that the audits be planned and performed. The results of audits are communicated to management, and any deficiencies found are corrected.

Specific audits in the CMM are called out in the auditing practices of the verifying implementation common feature.

#### **Q.72 Spiral model is a realistic approach to the development of large-scale systems & software. Justify & explain the model?**

**Ans.** There are several advantages of Spiral model that makes it a realistic approach to development of large-scale systems and software, viz:

- 1) The spiral model promotes quality assurance through prototyping at each stage in system development.
- 2) The spiral model is a realistic approach to the development of large-scale software products because the software evolves as the process progresses. The developer and client better understand and react to risk at each evolutionary level.
- 3) The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development.

4) It maintains a systematic stepwise approach like the classic life cycle model and incorporates it into an iterative framework that more reflects the real world.

If employed correctly this model should reduce risk before they become problematic as consideration of technical risk are considered at all stages

Spiral model is also known, as spiral life cycle model is a system development life cycle model used in information technology. This model of development combines the features of prototyping model and waterfall model. The spiral model is used for large, expensive and complicated project.

- Presented by BARRY BOHEM in 1986 incorporating the project risk factor.
- Designed in order to overcome the disadvantages of waterfall model.
- The radial dimension represents the cumulative cost incurred in accomplishing the steps done so far and the angular dimension represents the progress made in completing each cycle of the spiral.
- Each loop is a development stage.
- Balance all the risk elements that is the high-risk element must be lowered.
- The people concerned with the project complete each phase with a review.

#### Different Phases of spiral model:

1) **Planning:** In this phase the objectives, alternatives and constraints of the project are determined or documented. The objectives and other specifications are fixed in order to decide which strategies or approaches to follow during the project life cycle.

2) **Risk analysis:** In this phase, all possible and available alternatives which can help in developing a cost effective project are analyzed and strategies are decided to use them. This phase has been added specially in order to identify and resolve all the possible risk in the project development.

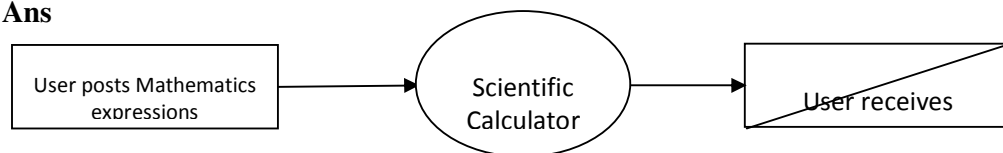
3) **Development:** In this phase the actual development of the project is carried out. The output of this phase is passed through all the phases iteratively in order to obtain improvements in the same.

4) **Assessment:** In this phase, developed product is passed on to the customer in order to receive customer comments and suggestions, which can help in identifying and resolving potential problems or errors in the software development.

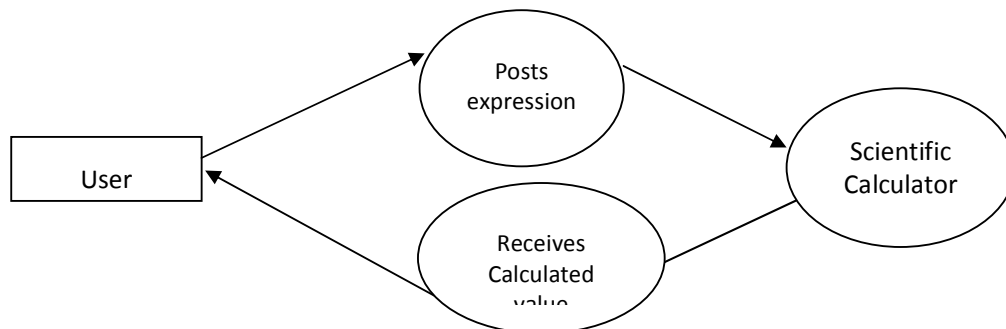
Q.73

**A program is to be developed to simulate the operations of a scientific calculator. List the facilities to be provided by this calculator. Analyze this using DFD?**

**Ans**



Context Free Diagram



**Q.74 What is function point? Explain its importance. What is function-oriented metrics?**

**Ans.** Function point measures the functionality from the user point of view, that is, on the basis of what the user request and receives in return. Therefore, it deals with the functionality being delivered, and not with the lines of code, source modules, files etc. measuring size in this way has the advantage that size measure is independent of the technology used to deliver the function.

**Importance of function point:**

- This is independent of the languages tools, or methodology used for implementation.
- They can be estimated from requirement specification or design specification.
- They are directly linked to the statement of request.

**Function-oriented metrics:** Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. function-oriented metrics were first proposed by Albrecht, who suggested a measure called function point. Function points are derived using an empirical relationship based on countable measures of software's information domain and assessments of software complexity.

**Q.75 Explain basic information flow model & its more sophisticated versions?**

**Ans.** Information flow metrics are applied to the components of a system design.

For any component 'A', we can define three measures:

- 1: 'FAN IN' is simply a count of the number of other components that can call, or pass control, to component A.
- 2: 'FAN OUT' is the number of components that are called by component A.
- 3: using the following formula derives this.

We will call this measure the INFORMATION FLOW index of component A, abbreviated as IF(A).

$$IF(A) = \{FAN\ IN(A) * FAN\ OUT(A)\}^2$$

The sophisticated IF model differs from basic model in its definition of FAN IN and FAN OUT

For a component A let:

a= the number of components that call A.

b=the number of parameters passed to A from components higher in the hierarchy

c= the number of parameters passed to A from components lower in the hierarchy

d=the number of data elements read by components A

then

$$FAN\ IN(A) = a + b + c + d$$

Also let:

e= the number of components called by A.

f=the number of parameters passed from A to components higher in the hierarchy

g= the number of parameters passed from A to components lower in the hierarchy

h=the number of data elements written to by A

FAN OUT(A)=e+f+g+h

**Other than those changes to the basic definitions, the derivation, analysis and interpretation remain the same.**

**Q.76 Explain COCOMO model with its relevant equations. Explain various attributes of cost drivers used in COCOMO model.**

**Ans.** COCOMO stands for constructive cost model

- An empirical model based on project experience.
- Well-documented, independent model, which is not tied to a specific software vendor.
- There is a long history of COCOMO model, from initial versions published in 1981(COCOMO-81) through various versions instantiations to COCOMO-2.
- Then COCOMO-2 takes into account different approaches to software development, reuse etc.

**Various attributes of cost drivers used in COCOMO model are:**

1. **Physical attributes:** These are concerned with required characteristics of the software product being developed.
2. **Computer attributes:** These are constraints imposed on the software by the hardware platform. These affect software productivity because effort must be expended to overcome the hardware limitations.
- 3: **Personnel attributes:** These are multipliers which takes into account the experience of the people working on the project.
- 4: **Project attributes:** These are concerned with the use of software tools, the project development schedule and the use of modern programming practices.

**Q.77 What is Data Binding?**

**Ans.**The matrix that attempts to capture the module-level concept of coupling is data binding. Data binding are a measure that captures the data interaction across portions of a software system. In other words, data binding try to specify how strongly coupled different modules in a software system are.

**Q.78 Define cohesion & coupling? Give suitable examples.**

**Ans Coupling:**Coupling refers to the strength of the relationship between modules in a system. Coupling represents how strongly different modules are interconnected with each other.

**Cohesion:** Cohesion refers to the strength of relationship among elements within a module. Cohesion represents how strongly the internal elements of a module are bound to each other.

**Q.79 Explain various Object Oriented concepts used in Software Engineering. Give an example.**

**Ans .Various concepts of Object Oriented concepts used in Software Engineering:**

- **Object:-** An object is something which is capable of being seen, touched or sensed. Each object has certain distinctions or attributes which

enable you to recognize and classify it. Each object has certain actions and methods associated with it.

- **Class:-**A class encapsulates data and procedural abstractions required to describe the content and behavior of some real world entity. A class is a generalized description that describes a collection of similar objects.
- **Encapsulation:-** An object is said to be encapsulate (hide) data and program. This means that the user cannot see the inside of the object but can use the object by calling the program part of the object. This hiding of details of one object from another object, which uses it, is known as encapsulation.
- **Inheritance:-**Inheritance is defined as the property of objects by which instances of a class can have access to data and programs contained in a previously defined class, without those definitions being restated. Classes are linked together in a hierarchy. They form a tree, whose root is the class of “objects”. Each class (except for the root class) will have a super class (a class above it in the hierarchy) and possibly subclasses. A class can inherit (i.e. acquire) methods from its super class and in turn can pass methods on to its subclasses. A subclass must have all the properties of the parent class, and other properties as well.
- **Polymorphism:-**Polymorphism includes the ability to use the same message to objects of different classes and have them behave differently. Thus we could define the message “+” for both the addition of numbers and the concatenation (joining) of characters. Polymorphism provides the ability to use the same word to invoke different methods, according to similarity of meaning.
- **Object/class associations:-**Objects/classes interact with each other. Multiplicity defines how many instances of one object/class can be associated with one instance of another object/class.
- **Messages:-**The interaction or communication between the different objects and classes is done by passing messages. The object, which requires communicating with another object, will send a request message to the latter. A message can be sent between two objects only if they have an association between them.

#### Q.80 . Compare and contrast reliability and availability?

**Ans.** Reliability:The probability of failure-free system operation over a specified time in a given environment for a given purpose is called reliability

Availability:The probability that a system, at a point in time, will be operational and able to deliver the requested services is known as availability.

Both of these attributes can be expressed quantitatively. It is sometimes possible to subsume system availability under system reliability. Obviously if a system is unavailable it is not delivering the specified system services. However, it is possible to have systems with low reliability that must be available. So long as system failures can be repaired quickly and do not damage data, low reliability may not be a problem.

Availability takes repair time into account.

#### Q.81 Design black box test suits for a function that checks whether a character or string upto ten characters in a palindrome?

**Ans** Following may be the test cases:

- Try with Even number of characters (MALAYALAM)
- Try with Odd number of characters (ADDA)
- Try with maximum length of string

The following are negative test cases: (Warning message should come for the following)

- Try with empty string
- Try with numbers
- Try with special characters

**Q.82 How is cyclomatic complexity useful in program test? What is sequence of testing? What is testability?**

**Ans.** Cyclomatic complexity measures the amount of decision logic in a single software module. It is used for two related purposes in the structured testing methodology. First, it gives the number of recommended tests for software. Second, it is used during all phases of the software lifecycle, beginning with design, to keep software reliable, testable, and manageable. Cyclomatic complexity is based entirely on the structure of software's control flow graph.

The sequence of testing is:

**Unit testing:** Unit testing is undertaken after a module has been coded and successfully reviewed. Unit testing (or module testing) is the testing of different units (or modules) of a system in isolation.

**Integration testing:** The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan. The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested.

**System testing** System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:

- **Alpha Testing-** Alpha testing refers to the system testing carried out by the test team within the developing organization.
- **Beta testing-** Beta testing is the system testing performed by a select group of friendly customers.
- **Acceptance Testing-** Acceptance testing is the system testing performed by the customer to determine whether he should accept the delivery of the system.

**Software testability** is the degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context.

Testability is not an intrinsic property of a software artifact and can not be measured directly. Instead testability is an extrinsic property which results from interdependency of the software to be tested and the test goals, test methods used, and test resources.

A lower degree of testability results in increased test effort. In extreme cases a lack of testability may hinder testing parts of the software or software requirements at all.

**Q.83 Explain various types of static and dynamic testing tools.**

**Ans. Static testing tools:**

1. **Static analysers** A static analyser operates from a pre-computed database of descriptive information derived from the source text of the program. The idea of a static analyser is to provide allegations, which are claims about the analysed programs that can be demonstrated by systematic examination of all cases.

2. **Code inspectors** A code inspector does a simple job of enforcing standards in a uniform way for many programs. These can be single statement or multiple statement rules. The AUDIT system is available which imposes some minimum conditions on the programs.

3. **Standard enforces** This tool is like a code inspector; expect that the rules are generally simpler. The main distribution is that a full-blown static analyser looks at whole programs, whereas a standard enforcer looks at only single statements.

**Dynamic testing tools**

1. **Coverage analyzers (execution verifiers)** A coverage analyzer is the most common and important tool for testing. It is often relatively simple. One of the common strategies for testing involves declaring the minimum level of coverage, ordinarily expressed as a percentage of the elemental segments that are exercised in aggregate during the testing process.

2. **Output comparators** These are used in dynamic testing-both single-module and multiple-module (system level) varieties to check that predicted and actual outputs are equivalent. This is also done during regression testing.

3. **Test data generators** This is a difficult one, and at least for the present is one for which no general solution exists. One of the practical difficulties with test data generation of sets of inequalities that represent the condition along a chosen path.

4. **Test file generators** This creates a file of information that is used as the program and does so based on comments given by the user and/or from the data descriptions program's data definition section.

5. **Test harness systems** This is one that is bound around the test object and that permits the easy modification and control of test inputs and output's and provides for online measurement of CI coverage values.

6. **Test archiving systems:** The goal is to keep track of series of tests ant to act as the basis for documenting that the tests have been done and that no defects were found during the process.

**Q. 84 Explain the following Software Metrics****(i) Lines of Code****(ii) Function Count****(iii) Token Count****(iv) Equivalent size measure**

**Ans (i) Lines of code (LOC)** is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. LOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced. **Advantages:-**

1. **Scope for Automation of Counting:** Since Line of Code is a physical entity; manual counting effort can be easily eliminated by automating the counting process. Small utilities may be developed for counting the LOC in a program. However, a code counting utility developed for a specific language cannot be used for other languages due to the syntactical and structural differences among languages.

2. **An Intuitive Metric:** Line of Code serves as an intuitive metric for measuring the size of software due to the fact that it can be seen and the effect of it can be

visualized. Function Point is more of an objective metric which cannot be imagined as being a physical entity, it exists only in the logical space. This way, LOC comes in handy to express the size of software among programmers with low levels of experience.

Disadvantages

1. **Lack of Accountability:** Lines of code measure suffers from some fundamental problems. Some think it isn't useful to measure the productivity of a project using only results from the coding phase, which usually accounts for only 30% to 35% of the overall effort.
2. **Lack of Cohesion with Functionality:** Though experiments have repeatedly confirmed that effort is highly correlated with LOC, functionality is less well correlated with LOC. That is, skilled developers may be able to develop the same functionality with far less code, so one program with less LOC may exhibit more functionality than another similar program. In particular, LOC is a poor productivity measure of individuals, because a developer who develops only a few lines may still be more productive than a developer creating more lines of code.
3. **Adverse Impact on Estimation:** Because of the fact presented under point (a), estimates based on lines of code can adversely go wrong, in all possibility.
4. **Developer's Experience:** Implementation of a specific logic differs based on the level of experience of the developer. Hence, number of lines of code differs from person to person. An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience does, though they use the same language.

**(ii) Function count:-** It measures the functionality from the user point of view, that is, on the basis of what the user requests and receives in return. Therefore it deals with the functionality being delivered, and not with the lines of code, source modules, files etc. measuring size in this way has the advantage that size measure is independent of the technology used to deliver the function.

**Features:-**

- Function point approach is independent of the language tools, or methodologies used for implementation.
- They can be estimated from requirement specification or design specification.
- They are directly linked to the statement of requirements.
- They are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

**(iii) Token count:-** A program is considered to be series of tokens and if we count the number of tokens, some interesting results may emerge. Tokens are classified as either operators or operands. All software science measures are functions of the counts of these tokens.

Variables, constants and even labels are operands. Operators consist of arithmetic symbols such as +, -, /, \* and command names such as "while", "for", "printf", special symbols such as : =, braces, parentheses, and even function names such as "eof".

The size of the vocabulary of a program, which consists of the number of unique tokens to build a program, is defined as:-

$$n = n_1 + n_2$$

Where, n : vocabulary of a program.

n<sub>1</sub>: number of unique operators



$n_2$ : number of unique operands.

(iv) **Equivalent size measure:-** Models makes no distinction between a new software and the portion being reused. The equivalent size measure  $S_e$ , is a function of  $S_n$  (new developed code) and  $S_u$  (reused code); means that the efforts required to develop the software with  $S_n$  and  $S_u$  is equivalent to the effort of developing a product with  $S_e$  “from scratch”. Sometimes there may be an undesirable situation in which it actually costs more to put together several existing program segments then it would to construct equivalent software from scratch.

Boehm has proposed a function for equivalent size measure as

$$S_e = S_n + (a/100)S_u$$

Where the adjustment factor ‘a’ is determined by the percentage of modification required of design(DM), of the code(CM), and of the efforts necessary for the integration of the modified code(IM).

**Q. 85 Explain incremental model? Define core product and detailed plan.**

**Ans.** The incremental model is proposed by D.L PARNAS. It is basically implemented by combining elements of linear sequential model and iterative prototyping model. Incremental development refer to the process of developing to completion only a part of the requirement at a time by selectively developing parts of the requirements, designing, coding and testing software that means these require functions, the software product may be built up incrementally to its final configuration. This approach model releases the product partially to the customer in the beginning and slowly at increased functionality to the system. That is why it is called incremental model.

The model prioritizes the system requirements and implements them and implements them in groups. Each new release of the system enhances the functionality of the previously released system thereby reducing the cost of project. In the first release on the functionality of the product is offered to the customer. In the second release, functionality A+ functionality B is offered. Finally, in release 3 functionality A, B and C are offered. Therefore with each release in addition to new functionality in the system, functionality of earlier release may also be enhanced.

**Advantages of incremental Model:-**

- As product is to be delivered in parts, total cost of the project is distributed.
- Limited number of persons can be put on to the project because work is to be delivering in parts.
- As development activities for next release and use of early version of product is done. Simultaneously, error if found can be corrected.
- Customers or end users get the chance to see the useful functionality early in SDLC.
- As a result, of end users feedback requirements for successive release become clearer.
- Testing also becomes easier.
- Risk of failure of product is decreased as user start using the product early.

**Q.86 What is Data Dictionary? Explain each component?**

**Ans** Data dictionary is a storehouse of data giving information about data. It is a list of terms and their definition for all data items and data files of a system. A data dictionary contains descriptions and definitions concerning the data structure, data elements, their interrelationships and other characteristics of a system.

**Objectives of Data dictionaries:-**

- 1) A standard definition of all terms in a system, that is each item of data is uniquely identified and defined.
- 2) Easy cross-referencing between sub-systems, programs and modules.
- 3) Simpler program maintenance.

**Data Items:**

There are three classes of data items in a data dictionary:-

- 1) Data element- It is the smallest unit of data which cannot be meaningfully decomposed further. e.g. Employee number etc.
- 2) Data structure- A group of data elements forms a data structure.
- 3) Data Flows and Data Stores- data Flows are data structures in motion whereas data stores are data structures at rest.

**Q.87 What specific languages can be used in SRS? What are the advantages of using these specific languages of SRS?**

**Ans.** Requirement specification necessitates the use of some specification language. The language should support the desired qualities of the SRS- modifiability, understandability, unambiguous, and so forth. The language should be easy to learn and use.

For ease of understanding a natural language might be preferable. Though formal notations exist for specifying specific properties of the system, natural languages are now most often used for specifying requirements. The overall SRS is generally in a natural language, and when flexible and desirable, some specifications in the SRS may use formal languages.

The major advantage of using a natural language is that both client and superior understand the language. However, by the very nature of a natural language, it is imprecise and ambiguous. To reduce the drawback of natural language, most often natural language is used in a structured fashion. In structured English, requirements are broken into sections and paragraphs. Each paragraph is then broken into sub paragraphs.

In an SRS, some parts can be specified better using some formal notation, example- to specify formats of inputs or outputs, regular expression can be very useful.

Similarly when discussing system like communication protocols, finite state automata can be used. Decision tables are useful to formally specify the behavior of a system on different combination of input or settings.

**Q.88 Draw the flow chart of Risk Management-Activity and explain various Software risks.**

**Ans** "RISK" is a problem that could cause some loss or threatens the success of the project, but which has not happened yet.

Typical Software Risks:-

1. Dependencies: -Many risks arise due to dependencies of project on outside agencies or factors. It is not easy to control these external dependencies. Some typical dependency-related risk factors are:-

- Availability of trained, experienced people.
  - Intercomponent or inter-group dependencies.
  - Customer-furnished items or information
  - Internal and external subcontractor relationships.
2. Requirement Issues:-Many project face uncertainty and turmoil around the product's requirements. If we do not control requirements-related risk factors, we might either build the wrong product, or build the right product badly. Either situation results in unpleasant surprises and unhappy customers. Some typical factors:-
- Lack of clear product vision.
  - Lack of agreement on product requirements.
  - Unprioritized requirements.
  - New market with uncertain needs.
  - Rapidly changing requirements.
3. Management Issues:-Project managers usually write the risk management plan, and most people do not wish to air their weaknesses in public. If we do not confront such touchy issues, we should not be surprised if they bite us at some point.
- Inadequate planning and task identification
  - Inadequate visibility into actual project status.
  - Unclear project ownership and decision making.
  - Unrealistic commitments made, sometimes for the wrong reasons.
  - Staff personality conflicts.
  - Poor communication.
4. lack of Knowledge:-The rapid rate of change of technologies, and the increasing change of skilled staff, means that our project teams may not have the skills we need to be successful. Some of the factors are:-
- Inadequate training.
  - Poor understanding of methods, tools, and techniques.
  - Inadequate application domain experience.
  - New technologies.
  - Ineffective, poorly documented, or neglected processes.

**Q.89 Software project planning entails what activities? What are the difficulties faced in measuring the Software Costs?**

**Ans.** Software project planning entails the following activities:

- Estimation:
  - –Effort, cost, resource, and project duration
- Project scheduling:
- Staff organization:
  - –staffing plans
- Risk handling:
  - -identification, analysis, and abatement procedures
- Miscellaneous plans:
  - –quality assurance plan, configuration management plan, etc.

Software costs are due to the requirement for software, hardware and human resources. One can perform cost estimation at any point in the software life cycle.

As the cost of software depends on the nature and characteristics of the project, the accuracy of estimate will depend on the amount of reliable information we have about the final product. So when the product is delivered, the costs can be actually determined as everything spend is known by then. However when the software is being initiated or during feasible study, we have only some idea about the functionality of software. There is very high uncertainty about the actual specifications of the system hence cost estimations based on uncertain information cannot be accurate.

**Q.90 Explain the types of COCOMO Models and give phase-wise distribution of effort.**

**Ans.** COCOMO model stand for Constructive Cost Model. It is an empirical model based on project experience. It is well-documented, independent model, which is not tied to a specific software vendor. Long history from initial version published in 1981(COCOMO-81) through various instantiations to COCOMO 2. COCOMO 2 takes into account different approaches to software development, reuse etc.

This model gives 3 levels of estimation namely basic, intermediate and detail.

1) Basic COCOMO model:- It gives an order of magnitude of cost. This model uses estimated size of software project and the type of software being developed.

The estimation varies for various types of projects and these various kinds are:-

- Organic-mode project:- These project include relatively small teams working in a familiar environment, developing a well understood application, the feature of such project are-

- 1) The communication overheads are low.
- 2) The team members know what they can achieve.
- 3) This project is much common in nature.

- Semi-detached model: A project consists of mixed project team of experienced and fresh engineers. The team has limited experience of related system development and some of them are unfamiliar with output and also some aspects of system being developed.

- Embedded model:- There will be very strong coupled hardware, software regulations and operational procedures. Validation costs are very high. For e.g. System program and development of OCR for English.

2) Intermediate COCOMO model:-

The intermediate COCOMO model estimates the software development effort by using 15 cost drivers' variables besides the size variable used in basic COCOMO.

3) Detailed COCOMO model:-

The detailed COCOMO model can estimate the staffing cost and duration of each of the development phases, subsystem, and modules. It allows you to experiment with different development strategies, to find the plan that best suits your needs and resources.

**Q.91 Differentiate between function oriented design and object oriented design.**

**Ans.** Function oriented design:- Function oriented design strategy relies on decomposing the system into a set of interacting functions with a centralized system state shared by these functions. Functions may also maintain local state information but only for the duration of their execution. Function oriented design

conceals the details of an algorithm in a function but system state information is not hidden.

**Object oriented design:-**Object oriented design transforms the analysis model created using object-oriented analysis into a design model that serves as a blueprint for software construction. It is a design strategy based on information hiding. Object oriented design is concerned with developing an object-oriented model of a software system to implement the identified requirements. Object oriented design establishes a design blueprint that enables a software engineer to define object oriented architecture in a manner that maximizes reuse, thereby improving development speed and end product quality.

**Object oriented design Vs function oriented design-**

- Unlike function oriented design methods, in OOD, the basic abstractions are not real world function such as sort, display, track etc. but real world entities such as employee, picture, machine etc.
- In OOD, state information is not represented in a centralized shared memory but is distributed among the objects of the system.

### **Q.92 What is stepwise refinement? Discuss partitioning & abstraction.**

**Ans.** Stepwise Refinement:-Stepwise Refinement is a top-down design strategy originally proposed by Niklaus Wirth. A program is developed by successively refining levels of procedural detail. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached.

Refinement is actually a process of elaboration. We begin with a statement of function that is defined at a high level of abstraction. That is, the statement describes function or information conceptually but provides no information about the internal workings of the function or the internal structure of the information. Refinement causes the designer to elaborate on the original statement, providing more and more detail as each successive refinement occurs.

**Partitioning:-**Problems are often too large and complex to be understood as a whole. For this reason, we tend to partition such problems into parts that can be easily understood and establish interfaces between the parts so that overall function can be accomplished.

Partitioning decomposes a problem into its constituent parts. We establish a hierarchical representation of function or information and then partition the uppermost element by:-

- 1) Exposing increasing detail by moving vertically in the hierarchy or
- 2) Functionality decomposing the problem by moving horizontally in the hierarchy.

**Abstraction:-** Abstraction permits one to concentrate on a problem at some level of generalization without regard to irrelevant low level details; use of abstraction also permits one to work with concepts and terms that are familiar in the problem environment without having to transform them to an unfamiliar structure.

- It allows considering the modules at the abstract level without worrying about its details.
- It provides external behavior of the modules.
- It is used for existing modules as well as for modules that are being design.
- It is essential for the problem partitioning.

**Q.93 Differentiate between failures and faults.**

**Ans.** Failure:-Failure is the departure of external results of program operation from requirements. So failure is something dynamic. Failure can also be defined as deficiency in performance, attributes and excessive response time.

There are four general ways of characterizing failure occurrence in time:-

- 1) Time of failure.
- 2) Time interval between failures.
- 3) Cumulative failures experienced up to a given time.
- 4) Failure experienced in a time interval.

Various failure classes are transient, permanent, recoverable, unrecoverable, non-corrupting, and corrupting.

Fault:- fault is the defect in the program that, when executed under particular condition, causes of failure. A fault is a property of the program rather than a property of its execution or behavior.

Various fault classes are data faults, control faults, input/output faults, interface faults, storage management faults, and exception management faults.

**Q.94 What do you understand by black box testing? Explain:**

**(i) Equivalence class**

**(ii) Equivalence partitioning**

**Ans.** Black Box Testing:-Black Box Testing is also called behavioral testing, focuses on the functional requirements of the software. It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. It is a complementary approach that is likely to uncover a different class of errors than white box testing.

Black Box Testing attempts to find errors in the following categories:-

- 1) Incorrect or missing functions.
- 2) Interface errors.
- 3) Errors in data structures or external database access.
- 4) Behavior or performance errors.
- 5) Initialization and termination errors.

Black Box Testing tends to be applied during later stages of testing because black box testing purposely disregards control structures; attention is focused on the information domain.

Equivalence class:-It represents a set of valid or invalid states for input conditions. An input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition. Equivalence class may be defined according to the following guidelines:-

- 1) If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
- 2) If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
- 3) If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
- 4) If an input condition is Boolean, one valid and one invalid class are defined.

Equivalence Partitioning:-Equivalence partitioning is black box testing method that divides the input domain of a program into classes of data from which test cases can

be derived. An ideal test case single-handedly uncovers a class of errors that might otherwise require many cases to be executed before the general error is observed. Equivalence partitioning strives to define test case that uncovers classes of errors, thereby reducing the total number of test case that must be developed.

### Q.95 Explain

(i) Reverse Engineering

(ii) Re-Engineering

**Ans i) REVERSE ENGINEERING:-**It is a process of analyzing software with a view to understanding its design and specification.

- In this, source code and executable code are the input.
- It may be part of a re-engineering process but may also be used to re-specify a system for re-implementation.
- Builds a program data base and generates information from this.
- Program understanding tools (browsers, cross reference generates, etc.) may be used in this process.
- Design and specification may be reverse re-engineer to:-
  - a) Serve as input to SRS for program replacement.
  - b) Be available to help program maintenance.

Reverse Engineering often precedes Re-Engineering but is sometimes worthwhile in its own right. The design and specification of a system may be reverse engineered so that they can be an input to the requirements specification process for the system replacement. The design and specification may be reverse engineered to support program maintenance.

ii) RE-ENGINEERING:- It is re-organizing and modifying existing system to make them more maintainable. It involves:-

- Source code translation.
- Reverse engineering.
- Program structure development.
- Program modularization.
- Data re-engineering.

Restructuring or re-writing part or all of the legacy system without hanging its functionality. Legacy system is a system that is hard to maintain. So it involves:-

- 1) Re-documenting the system.
- 2) Organizing and re-structuring the system.
- 3) Modifying and upgrading structure and value of the system data.
- 4) Input to a re-engineering process is a legacy system and output is a structure modularized version of the same program. So re-engineering involves adding effort to make them easier to maintain. The system may be restructured or redocumented.

When to Re-Engineer?

- When the system changes are mostly confined to part of the system then re-engineer that part.
- When hardware or software support becomes obsolete.
- When tools to support re-structuring are available.

Advantages of Re-Engineering:-

- 1) Reduced risk – there is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- 2) Reduced cost – the cost of re-engineering is often significantly less than the cost of developing new software.

Re-Engineering cost factors:-

- 1) The quality of the software to be re-engineered.
- 2) The tool support available for re-engineering.
- 3) The extent of the data conversion, which is required.
- 4) The availability of expert staff for re-engineering.

**Q.96 Explain various types of debugging techniques used in Software testing.**

**Ans.** Debugging is the activity of locating and correcting errors. Various debugging techniques are:-

1) Core dumps:-A printout of all registers and relevant memory location is obtained and studied. All dumps should be well documented and retained for possible use on subsequent problems.

**Advantages:-**

1. The complete contents of a memory at a crucial instant of time are obtained for study.
2. Can be cost effective if used to explore validity of a well formulated error hypothesis.

**Disadvantages:-**

1. Require some CPU time, significant input time, and much analysis time.
2. Wasteful if used indiscriminately.
3. Hexadecimal numbers are cumbersome to interpret and it is difficult to determine the address of source language variables.

2) Traces:-Printout contains only certain memory and register contents and printing is conditional on some event occurring. Typical conditionings are entry, exit, or use of-

- 1) A particular subroutine, statement, macro, or database;
- 2) Communication with a terminal, printer, disk, or other peripheral;
- 3) The value of a variable or expression; and
- 4) Timed actuations in certain real time system.

A special problem with trace programs is that the conditions are entered in the source language and any changes require a recompilation.

3) Print statements:-The standard print statement in the language being used is sprinkled throughout the program to output values of key variables.

**Advantages:-**

- 1) This is a simple way to test whether a particular variable changes, as it should after a particular event.
- 2) A sequence of print statements portrays the dynamics of variable changes.

**Disadvantages:-**

- 1) They are cumbersome to use on large programs.
- 2) If used indiscriminately they can produce copious data to be analyzed much of which are superfluous.



4) Debugging programs:-A program which runs concurrently with the program under test and provides commands to examine memory and registers, stop execution of the program at a particular point, search for references to particular constants, variables, and registers.

Advantages:-

- 1) Terminal oriented real time program.
- 2) Considerable flexibility to examine dynamics of operation.

Disadvantages:-

- 1) Generally works on a machine language program.
- 2) Higher-level language versions must work with interpreters.
- 3) More commonly used on microcomputers than large computers.

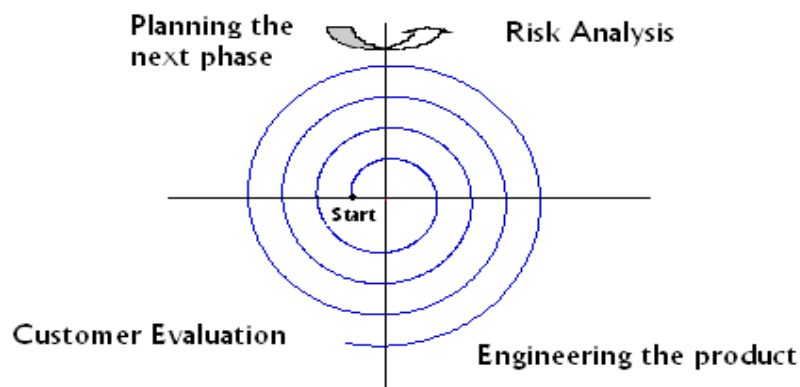
(ii) **Spiral Model:** The Spiral model is one of the popular model used for large projects. This model was proposed by Boehm in 1988 and it focuses on minimizing the risk through the use of prototype. We can view the Spiral Model as a waterfall model with each stage preceded by Risk analysis stage. The model is divided into four quadrants, each with a specific purpose as shown in the fig. Each spiral represents the progress made in the project. In the first quadrant objectives, alternative means to develop product and constraints imposed on the products are identified. The next quadrant deals with identification of risk and strategies to resolve the risks. The third bottom right quadrant follows the waterfall model. In the bottom left quadrant customer evaluates the product requirements are further refined. If at some stage during the project risk cannot be resolved, project is terminated. The model is used if the requirements are very complex or some new technology is being introduced by the company.

Advantages:

1. The model tries to resolve all possible risks involved in the project.
2. Each phase of the model improves the quality of the product.

Disadvantages:

1. The model is suitable only for large size projects because in some cases the cost of risk analysis may exceed the actual cost of the project.
2. Expertise in risk management and project management is essential.



**The Spiral Model**

Optimizing: This is the highest level of process maturity in CMM. This level focuses on continuous process improvement in the organization using quantitative feedback, latest tools and technology. The important KPAs at this level are:

- Defect Prevention
- Technology Change Management

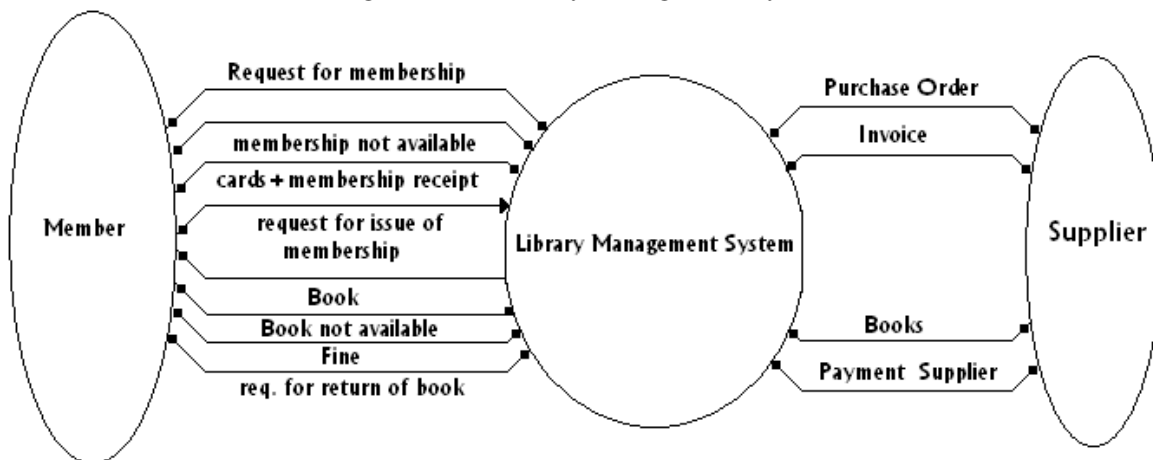
- Process Change Management

**Q.97 What is the use of a data flow diagram? Explain the important concepts of data flow diagram.**

**Ans.** A data flow diagram is used to show the functional view of an application domain. It shows all the important business processes and the flow of data between those processes. The main concepts used are:

- A process represents some kind of transformation on data. It takes as input one or more inputs and after doing necessary processing generates the output. It is represented by a circle with the name written inside as shown
- Data Flow: A data flow represents data in motion and is represented by an arrow. The data flows represent the flow of data among processes, stores and external agents.
- Data Store: A data store represents the data at rest. At the time of implementation it is represented by data base or files. Graphically it is shown as in figure.
- External Agent: An external agent represents a person, a system or any other software which interacts with the system by providing necessary inputs and outputs. Graphically it is represented by a rectangle.

(b) Draw a context diagram for a Library management system.



**Q.98 What is Software requirement Specification (SRS)? Why is it important? List the characteristic of a good quality SRS?**

**Ans:** Software Requirement Specification Document is the output of requirement analysis stage of the software development life cycle. It documents all types of requirements and constraints imposed on the end product. This document is important because it is used in all the successive stages of SDLC. Any error introduced here will result in to incomplete and bad quality product. The characteristics of a good quality SRS are:

- Correctness
- Completeness
- Consistency
- Unambiguousness
- Ranking for importance and/ or stability
- Modifiability
- Verifiability

- (viii) Traceability
- (ix) Design Independent
- (x) Understandable by customer

**Q.99 What is the difference between module coupling and module collection? List different types of coupling and cohesion.**

**Ans:** Cohesion is the property of a single module and can be described as a glue that keeps the data elements within a single module together. While defining, we must aim for high cohesion. Different types of cohesion are:

- Coincidental Cohesion
- Logical Cohesion
- Temporal Cohesion
- Communicational Cohesion
- Sequential Cohesion
- Functional Cohesion
- Procedural Cohesion

Coupling on the other hand is the measure of dependence among modules. A designer must try for minimum coupling. Different types of coupling are:

- Content Coupling
- Common Coupling
- Control Coupling
- Stamp Coupling
- Data Coupling

**Q 100 What are the different levels of testing? Explain in details.**

**Ans:**The different levels of testing are:

- Unit Testing
- Integration Testing
- System Testing

**Unit Testing:** The unit testing is done to test the individual module of the software. Test cases are designed to test the program logic, functionality, interfaces etc. in a module. Since individual module is being tested, white box testing techniques are used here. Wherever required, stubs and drivers are also written to test the module, which increase the cost of the testing.

**Integration Testing:** In this the module are systematically integrated and tested to find interface problems, protocol design error, errors due to global values, input/ output format errors etc. Different strategies can be:

- **Bottom up Integration:** In this type of testing the modules at the leaf level are first tested and then we move up in the hierarchy. Drivers are used in this type of testing at different levels of hierarchy. Driver is a program which accepts the test case data to be inputted and printed by the module to be tested.
- **Top Down Integration:** In this case we start the top most module in the hierarchy and move down till the leaf modules are tested. Where ever required necessary stubs/ drivers are used. A stub is a program which simulates the module called by the module to be tested.

- **Big-Bang Testing:** In this all the modules after unit testing are combined and tested in one go. The problem with this kind of testing is debugging of errors.
- **Sandwich Testing:** This technique makes use of combination of top down and Bottom up testing.

**System Testing:** This testing focuses on validating the product with respect to software specification Document. Techniques like function testing (using Black Box testing), Performance testing and acceptance testing (to be done by end user) is done here to test the product.

**Q.101 What are the advantages of using testing tools? Explain in detail different type of testing tools.**

**Ans:** The advantages of testing tools are:

- They improve the productivity and quality of software development.
- Help in identification of errors which are difficult and time consuming to find manually.
- Reduce the testing time.
- Help in running large volumes of test unattended for 24 hours.
- Automatic generation of test cases.
- Automated the regression testing.
- Improve the productivity of the testers.

**Some of the important testing tools are:**

(a) **Test Case Generators:** These tools generate test cases from SRS, program or test design languages. They use certain rules called test design techniques to generate test cases.

(b) **Capture/ Playback and Test harness tools:** These tools automate the re running of manual test by recording and replaying the test scripts. The recorded test scripts can also be edited as per need. They can be either intrusive or non intrusive type. Intrusive tools along with software under test reside on the same machine.

(c) **Coverage Analysis Tools:** These tools ensure that the software is tested and helps the tester to find the parts which are not covered.

(d) **Test Comparators:** These tools compare the results of software under test with the expected results and generate the report.

(e) **Memory Testing Tools:** These tools are used to test memory related problems. such as using uninitialized memory location, accessing memory locations which are out of range etc.

(f) **Simulators:** These tools are used to simulate the hardware/ software with which software under test is going to interact.

(g) **Test database:** It is a sample of database which is being manipulated by software under test.

**Q102 Define Reverse Engineering? What are the main objectives of reverse engineering?**

**Ans:** The reverse engineering is the process of generating representations that are implementation independent, starting from code. It is opposite of normal forward engineering process. The main objectives of the reverse Engineering process are:

- (i) It helps the companies to understand the complexities of the system
- (ii) Helps the analyst to generate useful lost information about legacy systems

- (iii) Can be used to identify reusable components for analysis and future use
- (iv) Helps in generating graphical representation of the system from different perspectives e.g. ER diagram, DFD, class diagram etc.
- (v) Can be used as a part of Reengineering process.
- (vi) Over a period of time modifications made to the software also result into unexpected problems. The anomalies can be detected using reverse engineering techniques.

**Q.103 Write short notes on following maintenance models – Quick-fix Model and Iterative Enhancement model.**

**Ans:** Quick-fix Model: This is the simplest model used for the maintenance of the software. In this model changes at the code level are made as early as possible without anticipating future maintenance problems. This model is not suitable for large and complex software systems. It should be used if the size of the software is small and is developed and maintained by one person. Also if the customers want fixing of bugs to be done immediately one can use this model. In that bugs can be fixed temporarily and parallelly, proper correction can be done.

Iterative Enhancement model: This model incorporates changes in the software based on the analysis of the existing system. Also the complete documentation of the system is available before doing any changes. In case of any change made, all the documents at different stages i.e. SRS, Design document, testing document etc. are also modified so as to support the next change successfully.

**Q.104 What is verification? (2)**

**Ans.** Verification is the process of determining whether the output of one phase of software development conforms to that of its previous phase, whereas validation is the process of determining whether a fully developed system conforms to its requirements specification. Thus while verification is concerned with phase containment of errors, the aim of validation is that the final product be error free.

**Q.105 Describe design walk throughs and critical design review. (8)**

**Ans.** A design walkthrough is a quality practice that allows designers to obtain an early validation of design decisions related to the development and treatment of content, design of the graphical user interface, and the elements of product functionality. Design walkthroughs provide designers with a way to identify and assess early on whether the proposed design meets the requirements and addresses the project's goal.

For a design walkthrough to be effective, it needs to include specific components. The following guidelines highlight these key components. Use these guidelines to plan, conduct, and participate in design walkthroughs and increase their effectiveness.

1. Plan for a Design Walkthrough
2. Get the Right Participants
3. Understand Key Roles and Responsibilities
4. Prepare for a Design Walkthrough
5. Use a Well-Structured Process
6. Review and Critique the Product, not the Designer

## 7. Review, do not Solve Problems

The purpose of critical design review is to ensure that the detailed design satisfies the specifications laid down during system design. The critical design review process is same as the inspection process in which a group of people gets together to discuss the design with the aim of revealing design errors or undesirable properties.

**Q.106 Explain the relationship between**

- (i) **Productivity and difficulty**  
 (ii) **Time and cost.** (6)

**Ans**

**(i) Productivity and difficulty**

**Productivity** refers to metrics and measures of output from production processes, per unit of input. Productivity P may be conceived of as a metrics of the technical or engineering efficiency of production. In software project planning, productivity is defined as the number of lines of code developed per person-month

**Difficulty** The ratio  $(K/t_d^2)$ , where K is software development cost and  $t_d$  is peak development time, is called difficulty and denoted by D, which is measured in person/year.

$$D = (K/t_d^2)$$

The relationship shows that a project is more difficult to develop when the manpower demand is high or when the time schedule is short.

Putnam has observed that productivity is proportional to the difficulty

$$P \propto D^\beta$$

The average productivity may be defined as

$$P = \text{Lines of code produced} / \text{Cumulative manpower used to produce code} \\ = S/E$$

Where S is the lines of code produced and E is cumulative manpower used from  $t=0$  to  $t_d$  (inception of the project to the delivery time)

**(ii) Time and cost**

In software projects, time cannot be freely exchanged against cost. Such a trade off is limited by the nature of the software development. For a given organization, developing software of size S, the quantity obtained is constant. We know

$$K^{1/3} t_d^{4/3} = S/C$$

If we raise power by 3, then  $Kt_d^4$  is constant for constant size software. A compression of the development time  $t_d$  will produce an increase of manpower cost.. If compression is excessive, not only would the software cost much more, but also the development would become so difficult that it would increase the risk of being unmanageable.

**Q 107 Write short notes on**

- (i) **Configuration Management**  
 (ii) **Decision Table**

**Ans: (i) Configuration Management:**

Due to several reasons, software changes during its life cycle. As a result of the changes made, multiple versions of the software exist at one time. These changes must be managed, controlled and documented properly in order to have reliable systems. Configuration management helps the developers to manage these changes systematically by applying procedures and standards and by using automated tools.

Though multiple versions of the software exist in the tool repository, only one official version of set of project components exist called baseline. The different components of the baseline are called configuration items. Some examples of configuration items are project plan, SRS, Design document, test plans, user manuals etc. A group of people constitute Configuration Control Board (CCB) which controls the changes to be made to the software. Whenever changes are to be made the following steps are followed:

- (i) Submit the change request along with details to CCB
- (ii) CCB accesses the change request after proper evaluation.
- (iii) Depending upon the results, the request is either accepted or rejected or can be deferred for the future assessment.
- (iv) If accepted, proper plan is prepared to implement the change.
- (v) Once changes are made, after validating by the quality personnel, all configuration items are updated.

Some popular configuration management tools are Clear CASE, Visual Source Safe etc.

**(ii) Decision Table:** When the process logic for a process involves multiple conditions and is very complicated, it is not advisable to use structured English. Instead decision tables are used. Main parts of the table are:

1. Condition Stubs
2. Action Stubs
3. Roles

Condition stubs list all the conditions relevant to the decision. Action part lists all the actions that will take place for a valid set of conditions. Finally rules part of the table specify the set of conditions that will trigger a particular action. A sample decision table is shown below:

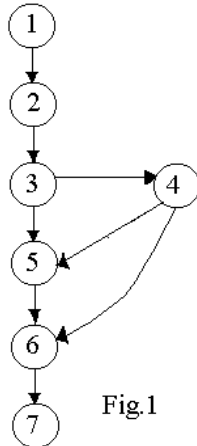
	Rule 1	Rule 2	Rule 3
Condition 1	X	X	
Condition 2		X	
Condition 3	X		X
Condition 4			X
Action 1	X		
Action 2		X	
Action 3			X

From this table it is clear that if conditions 1 and 3 are satisfied, Action 1 will be triggered.

## PART III

NUMERICALS

- Q 1** For the flow graph shown in fig ,  
 (i) compute the McCabe's cyclomatic complexity  
 (ii) Find out independent path



**Ans**

- (i) Cyclomatic complexity,  $V(G)$

$$V(G) = E - N + 2$$

Where E is the no of flow graph edges and N is the no of nodes .

So we have  $E = 8$   $N = 7$  and  $V(G) = 8 - 7 + 2 = 3$

- (ii) independent path

(i) 1,2,3,5,6,7

(ii) 1,2,3,4,5,6,7

(iii) 1,2,3,4,6,7

- Q 2** Drive an expression for peak manning of a project using Nordes / Rayleigh equation .

**Ans** Norden Rayleigh curve used to approximate software project .

$$M(t) = \text{Man power utilization} \\ = dy/dt = 2kae^{-at^2} \text{-----(1)}$$

K = Area under the curve  $t = \text{time}$

Integrating eq . (1) with respect to t

$$Y(t) = k[1 - e^{-at^2}] \text{-----(2)}$$

Differentiating eq (1) with respect to t

$$M'(t) = d^2y/dt^2 = 2kae^{-at^2}(1 - 2at^2) \text{-----(3)}$$

Put  $a = 1/2t_d^2$  and  $t = t_d$  in eq (1)

$$M(t) = 2kae^{-at^2} = 2k * 1/2t_d^2 * t^d e^{-1/2t_d^2 * t^d} \\ = k/t_d e^{-1/2} = k/t_d e^{1/2}$$

$$M_0 = k/t_d e^{1/2}$$

$M_0 = \text{peak manning of a project .}$



- Q 3** Assuming the Putnam model and given  $S = 100,000$   $C = 5000$  and  $D_0 = 15$  calculate the development time  $t_d$ . (4)

**Ans:** In Putnam model

$$(S/C)^3 = D_0 t_d^7$$

$$\text{lhs} = 20^3 = 8000$$

$$t_d^7 = 8000 / 15$$

$t_d$  is seventh root of 533.33

- Q 4.** Define the failure intensity of the Basic model. (4)

**Ans:** The failure intensity of the Basic model is

$$\lambda(\mu) = \lambda_0 [1 - \mu / v_0]$$

where  $\lambda_0$  is the initial failure intensity and  $\mu$  is the average number of failures expected at any given point in time. The quantity  $v_0$  is the total number of failures that would occur at infinite time.

- Q 5.** Assume a program will experience a total of 200 failures. Initial failure intensity is 16 failure/ CPU hr. It has now experienced 50 failures. Determine the following after specifying the formula
- Current failure intensity
  - Decrement of failure intensity
  - Failure intensity at 100 CPU hr. (4 x 3)

**Ans:**

- i. Current failure intensity

$$\lambda(\mu) = \lambda_0 [1 - \mu / v_0] = 16 [1 - 50 / 200] = 12 \text{ failure / CPU hr}$$

- ii. Decrement of failure intensity

$$d\lambda / d\mu = -\lambda_0 / v_0 = 16 / 200 = -0.08 \text{ CPU hr}$$

- iii. Failure intensity at 100 CPU hr

$$\begin{aligned} \lambda(\tau) &= \lambda_0 \exp(-\lambda_0 \tau / v_0) \\ &= 16 \exp(-16 * 100 / 200) = 16 \exp(-8) \end{aligned}$$

- Q 6** Consider the program given below

```
void main()
{
    int i,j,k;
    readln (i,j,k);
    if( (i < j) || (i > k) )
    {
        writeln("then part");
        if (j < k)
            writeln ("j less then k");
        else writeln ( " j not less then k");
    }
    else writeln( "else Part");
}
```

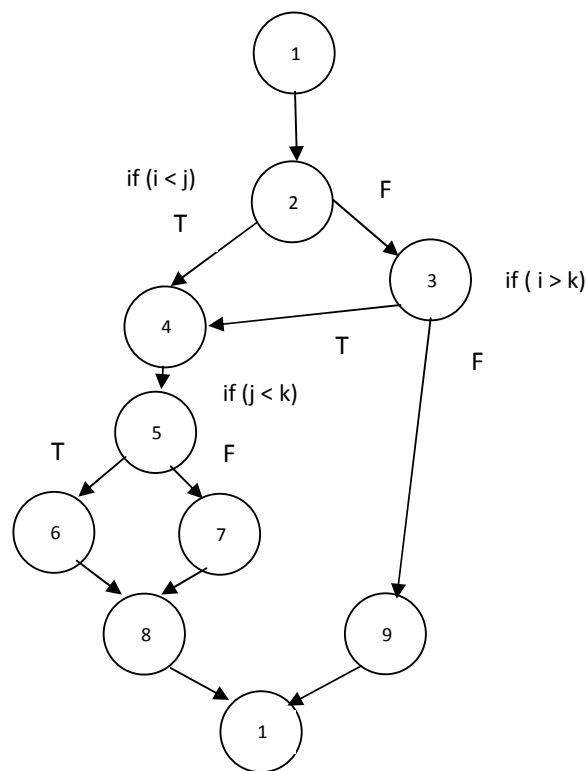
- Draw the flow graph. (4)
- Determine the cyclomatic complexity. (4)
- Arrive at all the independent paths. (8)

**Ans:**

```

void main()
{
1  int i,j,k;
2  readln (i,j,k);
3  if( (i < j) || (i > k) )
   {
4      writeln("then part");
5      if (j < k)
6          writeln ("j less then k");
7          else writeln ( " j not less then k");
8  }
9  else writeln( "else Part");
}

```



(ii) Cyclomatic complexity =  $E - N + 2 = 12 - 10 + 2 = 4$

(iii) The four independent paths are

Path1 : 1 2 3 9 10

Path2 : 1 2 4 5 7 8 10

Path3 : 1 2 4 5 6 8 10

Path4 : 1 2 3 4 5 7 8 10

- Q 7.**
- Define Annual Change Traffic (ACT) and Annual Maintenance Effort (AME) of Boehm model. (4)
  - For a software system of 90 KLOC, 5KLOC of code was added. The initial development cost was Rupees 8 lakhs with a total KLOC of 50. Total

lifetime for the software is 10 years. Compute ACT and the total cost of the system. (6)

**Ans: a)** Boehm proposed a formula for estimating maintenance costs as part of his COCOMO model. He used a quantity called Annual Change Traffic (ACT) which is defined as:

“ The fraction of a software product’s source instructions which undergo change during a year either through addition, deletion or modification

ACT is defined as

$$\text{ACT} = (\text{KLOC}_{\text{added}} + \text{KLOC}_{\text{deleted}}) / \text{KLOC}_{\text{total}}$$

(Annual Maintenance Effort)

AME is defined as ACT X SDE

where SDE is the software development effort in man-months]

ACT: Annual Change Traffic.

**b)**  $\text{ACT} = (5) / 50 = 0.10$

Development cost = Rs. 8 lakhs.

Life time = 10 years

$$\begin{aligned} \text{Total cost} &= \text{development cost} + 10 * (\text{development cost} * 0.10) \\ &= 8 + 10 * .8 = 16 \text{ lakhs} \end{aligned}$$

**Q 8.** Consider a project for which the manpower requirement is 200 PY and the development time is 2 years. calculate the peak manning time.

**Ans** Software Project cost  $K=200$  PY

Peak development time  $t_d = 2$  year = 2 years

Peak manning  $m_0 = (k / t_d * \text{sqrt}(e)) = 200/2 * 1.648 = 164.8 \approx 164$  persons

**Q.9** For the program given below, calculate the values of software science measures like  $\eta$ , N, V,E and  $\lambda$ .

1.	int. sort (int x[], int n)
2.	{
3.	int i, j, save, im1;
4.	/*This function sorts array x in ascending order*/
5.	If (n<2) return 1;
6.	for (i =2; i<=n;i++)
7.	{
8.	im1=i-1;
9.	for (j=1;j<=im;j++)
10.	if (x[i]<x[j])
11.	{
12.	Save = x[i];
13.	x [i]=x[j];
14.	x [j]=save;
15.	}
16.	}
17.	return 0;
18.	}

**Ans:**

This list of operators and operands is given in Table.

Operators	Occurrences	Operands	Occurrences
int	4	SORT	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3
;	11	im1	3
for	2	2	2
=	6	1	3
-	1	0	1
<=	2	-	-
++	2	-	-
return	2	-	-
{ }	3	-	-
$\eta_{1=14}$	$N_1=53$	$\eta_{2=10}$	$N_2=38$

Here  $N_1=53$  and  $N_2=38$ . The program length  $N = N_1 + N_2 = 91$

Vocabulary of the program  $\eta = \eta_1 + \eta_2 = 14 + 10 = 24$

Volume  $V = N \times \log_2 \eta$   
 $= 91 \times \log_2 24 = 417$  bits.

The estimated program length  $N$  of the program  
 $= 14 \log_2 14 + 10 \log_2 10$   
 $= 14 * 3.81 + 10 * 3.32$   
 $= 53.34 + 33.2 = 86.45$

Conceptually unique input and output parameters are represented by  $\eta^*$ .  
 $\eta^*_2 = 3$  {x: array holding the integer to be sorted. This is used both as input and output}.

{N: the size of the array to be sorted}.

The potential volume  $V^* = 5 \log_2 5 = 11.6$

Since  $L = V^*/V$   
 $= 11.6/417 = 0.027$   
 $D = 1/L$   
 $= 1/0.027 = 37.03$

Estimated program level

$$\hat{L} = \frac{2}{\eta_1} \times \frac{\eta_2}{N_2} = \frac{2}{14} \times \frac{10}{38} = 0.038$$

We may use another formula

$$\hat{V}^* = V \times \hat{L} = 417 \times 0.038 = 15.67$$

The discrepancy between  $V^*$  and  $\hat{V}^*$  does not inspire confidence in the application of this portion of software science theory to more complicated programs.

$$E = V / \hat{L} = \hat{D} \times V$$

$$= 417 / 0.038 = 10973.68$$

Therefore, 10974 elementary mental discriminations are required to construct the program.

$$T = E / \beta = \frac{10974}{18} = 610 \text{ seconds} \approx 10 \text{ minutes}$$

- Q.10** A software project is planned to cost 95PY in a period of 1 year and 9 months. Calculate the peak manning and average rate of software them build up. (5)

**Ans.** Software project cost  $K=95\text{PY}$   
 Peak development time  $t_d=1.75\text{years}$   
 Peak manning  $= m_o = \frac{K}{t_d \sqrt{e}}$

$$\frac{95}{1.75 \times 1.648} = 32.94 = 33 \text{ persons}$$

Average rate of software team build up  
 $= m_o / t_d = 33 / 1.75 = 18.8 \text{ person / year or } 1.56 \text{ person / month.}$

- Q.11** Assume that a program will experience 200 failures in infinite time. It has now experienced 100. The initial failure intensity was 20 failures/CPU hr. (6)
- Determine the current failure intensity
  - Find the decrement of failure intensity per failure.

**Ans:** Here

$$V_o = 200 \text{ failure}$$

$$\mu = 100 \text{ failure}$$

$$\lambda_o = 20 \text{ failures / CPU hr.}$$

- (i) Current failure intensity:

$$\lambda(\mu) = \lambda_o \left( 1 - \frac{\mu}{V_o} \right)$$

$$= 20 \left( 1 - \frac{100}{200} \right) = 20(1 - 0.5) = 10 \text{ failures / CPU hr.}$$

- (ii) Decrement of failure intensity per failure can be calculated as:

$$\frac{d\lambda}{d\mu} = \frac{-\lambda_o}{V_o} = -\frac{20}{200} = -0.1 / \text{CPU hr.}$$

- Q.12** Compute function point value for a project with the following domain characteristics:

No. of I/P = 30  
 No. of O/P = 62  
 No. of user Inquiries = 24  
 No. of files = 8  
 No. of external interfaces = 2

Assume that all the complexity adjustment values are average. Assume that 14 algorithms have been counted.

**Ans.** We know

$$UFP = \sum W_{ij} Z_{ij} \text{ where } j=2 \text{ because all weighting factors are average.}$$

$$= 30*4 + 62*5 + 24*4 + 8*10 + 2*7$$

$$= 120 + 310 + 96 + 80 + 14$$

$$= 620$$

$$CAF = (0.65 + 0.01 \sum F_i)$$

$$= 0.65 + 0.01(14*3)$$

$$= 0.65 + 0.42$$

$$= 1.07$$

$$nFP = UFP * CAF$$

$$= 620 * 1.07$$

$$= 663.4 \approx 663$$

**Q.13** Consider a program that reads a set of Data for 'n' no. of triangles. The program reads three integer values as representing the sides of triangles. The program prints for each triangle whether the triangle is isosceles or equilateral or a simple. Develop logic and do the following:

(i) Compute cyclomatic complexity?

(ii) Design test cases for loop testing?

**Ans.** The program logic will be as follows:

Enter three sides of a triangle.

Read a, b and c

If(a<b+c)AND(b<a+c)AND(c<a+b)

Then is\_a\_triangle=TRUE

Else is\_a\_triangle=FALSE;

IF is\_a\_triangle

Then

If(a=b)XOR(a=c)XOR(b=c) AND NOT ((a=b)AND(a=c))

Then print "Triangle is Isosceles"

If(a=b)AND(b=c)

Then print "Triangle is Equilateral"

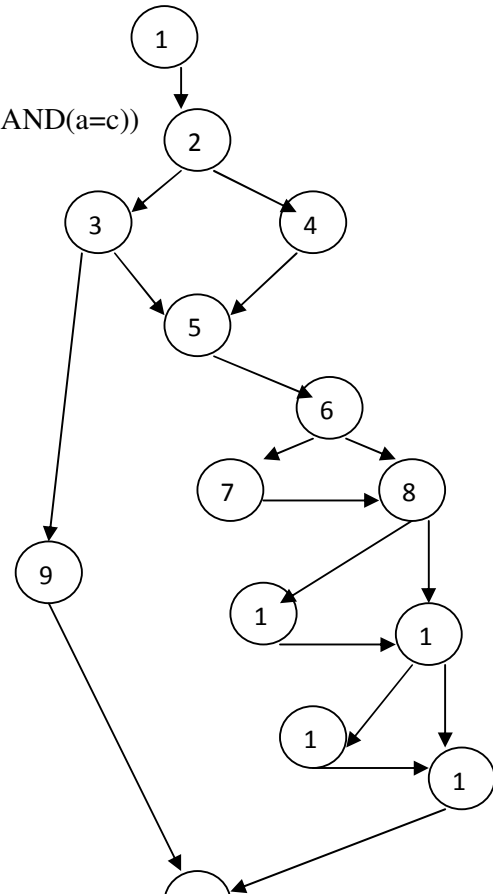
If(a<>b)AND (a<>c) AND(b<>c)

Then print "Triangle is scalene"

Else

Print "Not a triangle"

The flow graph of the problem is given as in the fig:



(i) There are 6 independent paths i.e.

1,2,3,9,14

1,2,4,5,6,8,11,13,14

1,2,3,5,6,8,11,13,14

1,2,4,5,6,7,8,11,13,14

1,2,4,5,6,8,10,11,13,14

1,2,4,5,6,8,11,12,13,14

Cyclomatic complexity is 6

(ii) Few test cases are:

Test #	Description	Test Data
1	A valid scalene triangle.	a = 2, b = 3, c = 4
2	An illegal triangle with a zero length side.	a = 0, b = 1, c = 1
3	An illegal triangle with one short side.	a = 1, b = 4, c = 2
4	A valid equilateral triangle.	a = 1, b = 1, c = 1
5	A valid isosceles triangle.	a = 1, b = 2, c = 2

**Q.14** Consider a program which computes the square root of an input integer between 0 and 5000. Determine the equivalence class test cases. Determine the test cases using boundary value analysis also.

**Ans.** For a program that computes the square root of an input integer which can assume values in the range of 0 to 5000, there are three equivalence classes: The set of negative integers, the set of integers in the range of 0 and 5000, and the integers larger than 5000. Therefore, the test cases must include representatives for each of the three equivalence classes and a possible test set can be: {-5,500,6000}.

Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes. For a function that computes the square root of integer values in the range of 0 and 5000, the test cases must include the following values: {0, -1,5000,5001}.

**Q.15** For the structural chart given in fig.1 calculate the Information Flow index of individual modules as well as whole software.

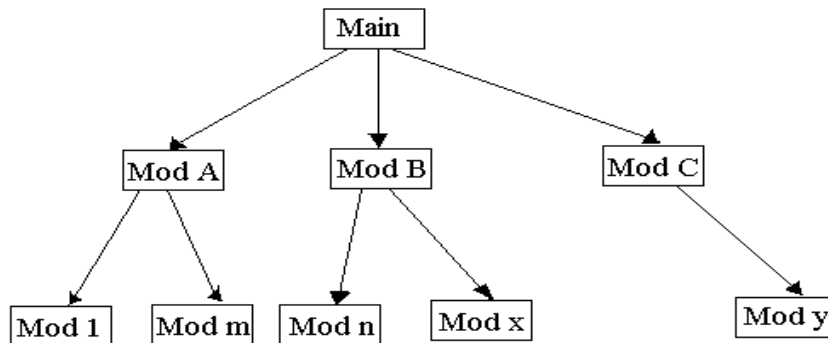


Fig.1

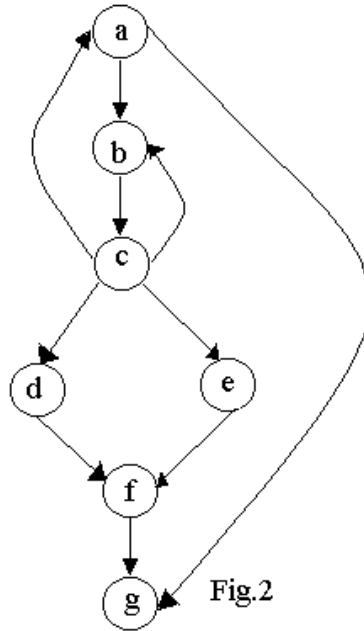
**Ans:**

Module	Fan-in	Fan-out	Information Flow index of module $(\text{Fan-in} * \text{Fan-out})^2$
main	2	2	16
A	2	2	16
B	2	2	16
C	2	1	4
l	0	1	0
m	1	1	1
n	0	1	0
x	1	1	1

y	1	0	0
---	---	---	---

$$\begin{aligned}
 \text{Total information flow} &= \text{IF (main)} + \text{IF (A)} + \text{IF (B)} + \text{IF (C)} + \text{IF (I)} + \text{IF (m)} \\
 &\quad + \text{IF (n)} + \text{IF (x)} + \text{IF (y)} \\
 &= 16 + 16 + 16 + 4 + 0 + 1 + 0 + 1 + 0 \\
 &= 54
 \end{aligned}$$

**Q.16** For the flow graph shown in Fig2, compute McCabe's Cyclomatic Complexity.



Ans: In this flow graph, Number of regions are 5, hence the Cyclomatic complexity is = 5

OR

No of edges = 10

No of nodes (N) = 7

Hence Cyclomatic complexity + E-N+2 = 10-7+2 = 5

**Q 17** Write a program in C to add two numbers and then calculate the values of the following software science metrics :

- (i) n(vocabulary of a program )
- (ii) N (program length)
- (iii) V (Volume)
- (iv) L (program level)

**Ans**            Void Main()

```

{ int i,j,sum
scanf("%d%d",i,j);
sum = i+j;
printf("%d" ,sum);}

```

operators	occurrences	operands	occurrences
main()	1		
		i	2



		j	2
scanf	1		
		sum	2
printf	1		
n1=3	N1=3	n2=3	N2=6

- (i)  $n(\text{vocabulary of a program}) = n_1 + n_2 = 6$
- (ii)  $N(\text{program length}) = N_1 + N_2 = 9$
- (iii)  $V(\text{Volume}) = N \log_2 n$   
 $= 9 \log_2 6$
- (iv)  $L(\text{program level}) = \frac{2}{n_1} * \frac{n_2}{N_2}$   
 $= \frac{2}{3} * \frac{3}{6}$   
 $= \frac{1}{3}$   
 $= 0.33$