UNIT-IV

Software Reliability and Quality Assurance :Quality concepts, Software quality assurance , SQA activities; Software reviews: cost impact of software defects, defect amplification and removal; formal technical reviews: The review meeting, review reporting and record keeping, review guidelines; Formal approaches to SQA; Statistical software quality assurance; software reliability: Measures of reliability and availability ,The ISO 9000 Quality standards: The ISO approach to quality assurance systems, The ISO 9001 standard, Software Configuration Management. Computer Aided software Engineering: CASE, building blocks, integrated case environments and architecture, repository.

NOTE : Examiner will set nine questions in total. Question one will be compulsory. Question one will have 6 parts of 2.5 marks each from all units and remaining eight questions of 15 marks each to be set by taking twoquestions from each unit. The students have to attempt five questions in total, first being compulsory and selecting one from each unit.

PRINCIPLES OF SOFTWARES

May - 2014
Paper Code:-CSE-302-F

Note: *Question No. 1 is compulsory, and attempt one question from each of the four sections. All questions carry equal marks.*

Q.1(a) What is software engineering ? (2.5)

Ans. IEEE Comprehensive Definition : Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software.

According to Barry Boehm : Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation.

Q.1(b) What is project metrics? (2.5)

Ans. Project Meterics : Project Metrics describe the project characteristics and execution. Examples are :

- Number of software developers
- Staffing pattern over the life cycle of the software
- Cost and schedule
- Productivity

Q.1(c) What is data dictionary? (2.5)

Ans. Data Dicitionary : "The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of input, outputs, and components of stores, and even intermediate calculations."

A data dictionary is like any other dictionary, in that it defines each data element name. The data dictionary contains definitions for all data elements in the system being modelled, no more and no less.

Q.1(d) What is information hiding ? (2.5)

Ans. Information hiding is the process of hiding the details of an object or function. The hiding of these details results in an abstraction, which reduces the external complexity and makes the object or function easier to use. In addition, information hiding effectively decouples the calling code from the internal workings of the object or function being called, which makes it possible to change the hidden portions without having to also change the calling code. Encapsulation is a common technique programmers use to implement information hiding.

Q.1(e) What is forward engineering ? (2.5)

Ans. The second level of the software re-engineering process is referred to as "forward engineering". The forward engineering procedure corresponds to the customary procedures of the system generation.

The objective of forward engineering is to modify the software part of existing IT systems until they are really state-of-the-art, which (contrary to reverse engineering) may also include functional software modifications.

Typical examples for such modifications are :

- Use of a different Programming language,
- Introduction of a new database management system or
- Transfer of sofware to a new hardware platform.

Q.1(f) What is recover testing? **(2.5)**

Ans. Recovery testing uses test cases designed to examine how easily and completely the system can recover from a disaster (power shut down, blown circuit, disk crash, interface failure, insufficient memory, etc). It is desirable to have a system capable of recovering quickly and with minimal human intervention. It should also have a log of activities happening before the crash (these should be part of daily operations) and a log of messages during the failure (if possible) and re-start.

Q.1(g) Define CASE tool. **(2.5)**

Ans. CASE Tools are used to assist software engineering activities (like analysis modeling, code generation etc.) either communicating with other tools, project database (integrated CASE environment) or as point solutions.

Q.1(h) What is FTR ? **(2.5)**

Ans. Formal Technical Review (FTR) are formal examinations of software products to identify faults (i.e. departures from specifications and standards). They are a class of reviews that include:

(i) System definition Reviews
(ii) Design Reviews
(iii) Walk throughs
(iv) Inspections
(v) Test Readiness Reviews
(vi) Functional & Physical configuration Audits
(vii) Formal qualification reviews
(viii) Round-Robin Reviews
(ix) Other small group technical assesment of software

Section - A

Q.2(a) What is software crisis. State its significance in reference to software engineering discipline. **(10)**

Ans. Software crisis : The *software crisis* is characterized by *an inability to develop software on time, on budget and within requirements*. The software crisis has been with us since 1970s. As per the latest IBM report, :31% of the projects get cancelled before they are completed, 53% over-run their cost-estimates by an average of 189% and for every 100 projects, there are 94 restarts".

When software is developing then during development many problems are raised up that set of problem is known as software crists. When software is developing then on the different steps of development, problems are encountered associated with those steps. Now we will discuss the problems and causes of software crisis encounter on different stage of software development.

Problems :

- Schedule and cost estimates are often grossly inaccurate.
- The "Productivity" of software people hasn't kept pace with the demand for their services.
- The quality of software is sometimes less than adequate.
- With no solid indication of productivity, we can't accurately evaluate the efficiency of new, tools, methods or standards.
- Communication between customer and software developer is often poor.
- The software maintenance task devours the majority of all software rupees.

Causes:

- Quality of software is not good because most of the developer use the historical data to develop the software.
- If there is delay in any process or stage (i.e., analysis, design, coding & testing) then scheduling does not match with actual timing.
- Communication between managers and customers, software developers, support staff etc. can break down because the special characteristics of software and the problems associated with its development are misunderstood.
- The software people responsible for tapping that potential often change when it is discussed and resist change when it is introduced.

Software crisis in the Programmer's Point of view :

- Problem of compatibility.
- Problem of portability.
- Problem in documentation.
- Problem of piracy of software.
- Problem in co-ordination of work of different people.
- Problem in maintenance in proper manner.

Software crisis in the User's Point of View:

- Software cost is very high.
- Customers are moody or choosy.
- Hardware goes very down.
- Lack of specialization in development.
- Problem of different versions of software.
- Problem of views.
- Problem of bugs.

Q.2(b) What are the different risk management activites? Explain. (10)

Ans. The risk management process has several activities that are illustrated in Fig.

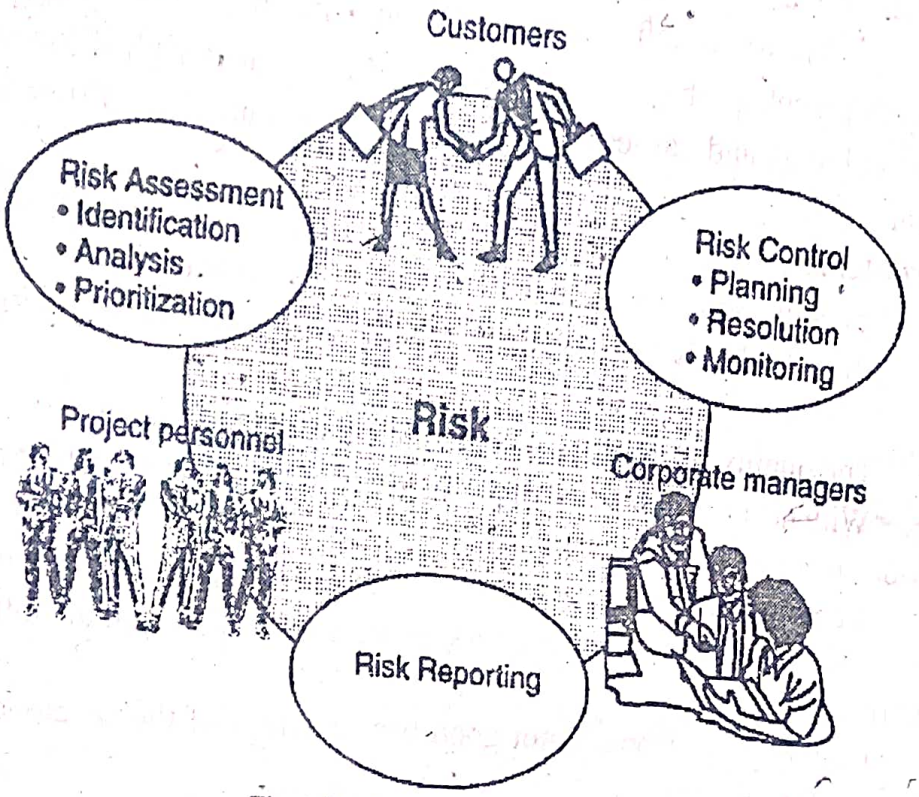


Fig. : Risk management Activities

(1) Risk Assessment : Risk assessment activity includes the following :

- *Risk Identification*
- *Risk Analysis*
- *Risk Prioritization*

(i) *Risk Identification :* Risk identification is a systematic attempt to specify threats the project plan. The purpose of *risk identification* is to develop a list of risk items called r statement. *Risk identificaion* can be facilitated with the help of a checklist of common r areas for software projects, or by examining the contents of an organizational databse of pre ously identified risks and mitigation strategies (both successful and unsuccessful).

Within the identification phase, several activities occur . The main activities are:

(a). Identify risks : There are many techniques to be used to identify risk. Some of the are check-lists, interviews, brainstorm meetings, reviews and surveys. A checklist to be used a tool for identification of risks is provided.

(b). Define risk attributes : After the risks are identified, they are evaluated with criteria: likelihood of occurrence (probability), consequence and time frame for action. Th values are initial estimations which are analysed more in the next phase.

(c). Document : The risks are then documented. Together with the name of the risk risk statement and context are to be specified. In this initial phase the description of the r issue, the probabilty and the consequence are specified in subjective terms.

(d). Communicate : Spreading the knowledge to the project members.

(ii) Risk Analysis : When the risks have been identified, all items are analyzed us different criteria. The purpose of the risk analysis is to assess the loss probability and magnitu of each risk item.

The input is the risk statement and context developed in the identification phase. T output of this phase is a risk list containing relative of the risks and a further analysis of

description, probability, consequence and context.

(iii) *Risk Prioritization : Risk prioritization* helps the project focus on its most severe risks by assessing the risk exposure. Exposure is the product of the probability of incurring a loss due to the risk and the potential magnitude of that loss.

(2) Risk Control : *Risk control* is the process of managing risks to achieve the desired outcomes. Risk control process involves the following activites :

(i) Risk Planning

(ii) Risk Mitigation

(iii) Risk Resolution

(iv) Risk Monitoring

(i) *Risk Planning :* Risk planning is to identify strategies to deal with risk. These strategies fall into three categories :

(a) Risk Avoidance

(b) Risk Minimization

(c) Risk Contingency Plans

(ii) *Risk Mitigation :* The risk mitigation is a plan that would reduce or eliminate the highest risks. The key question is: *What should be done and who is responsible to eliminate or minimize the risk?* The mitigation plan include a description of the action that can be taken to mitigate the red rated risk and assigns a primary handler for the action.

(iii) *Risk Resolution :* When a risk has occurred, it has to be solved. *Risk resolution* the execution of the plans for dealing with each risk. If the risk is at the watch list, a plan of how to resolve the risk already had taken place. The project manager has to respond to the already chalked out plan of how to resolve the risk.

(iv) *Risk Monitoring :* Risk Monitoring is the continually reassessing of risk as the project proceeds and conditions changes. For example, successful completion of beta testing means that the risk of the client organization rejecting the system is minimal, while large turnover in development staff usually increases project and product risks.

(3) Risk Reporting: Risk Reporting is reporting the status of the risks that were identified during risk identification and assessment stages.

Q.3(a) What do you understand by project scheduling ? Also enumerate the activities involved in project scheduling. (10)

Ans. Software Project Scheduling : Software Project Scheduling is one of the most difficult jobs of the software project manager. It is an extremely difficult because much of the information used in it is empirical and based on a particular individual's experience.

There are many techniques for estimating the time and resources required for a project (Mall 1999, Sommerville 1998, Pressman 1997). Estimates of the time and resources required for a development project are almost always optimistic, especially when the system being developed is technologically advanced or is significantly different from other projects that the organization has developed.

Additionally, competition for bids can lead to optimistic estimates. This is not unique to software development projects, but is more frequently true for software projects than other kinds of engineering projects.

Project Scheduling Activities : Project scheduling involves the following activities :

- Identify all the tasks and activities within the project
- Identify dependencies among the activities i.e. a unit must be tested before it can be integrated into the system.
- Estimate the resources (time) required for each activity
- Allocate people to activities
- Estimate the resources (time) required for the entire development project.

Q.3(b) Explain iterative enhancement model of S/W development life cycle.(10)

Ans. Iterative Enhancement Model : The Iterative enhancement model combines elements of the linear sequential model(applied repetitively) with the iterative philosophy of a prototyping. In this model, the software is broken down into several modules; which are incrementally developed and delivered. First, the development team develops the core module of the system and then it is later refined into increasing levels of capability of adding new functionalities in successive version. Each linear sequence produces a deliverable *increment* of the software.

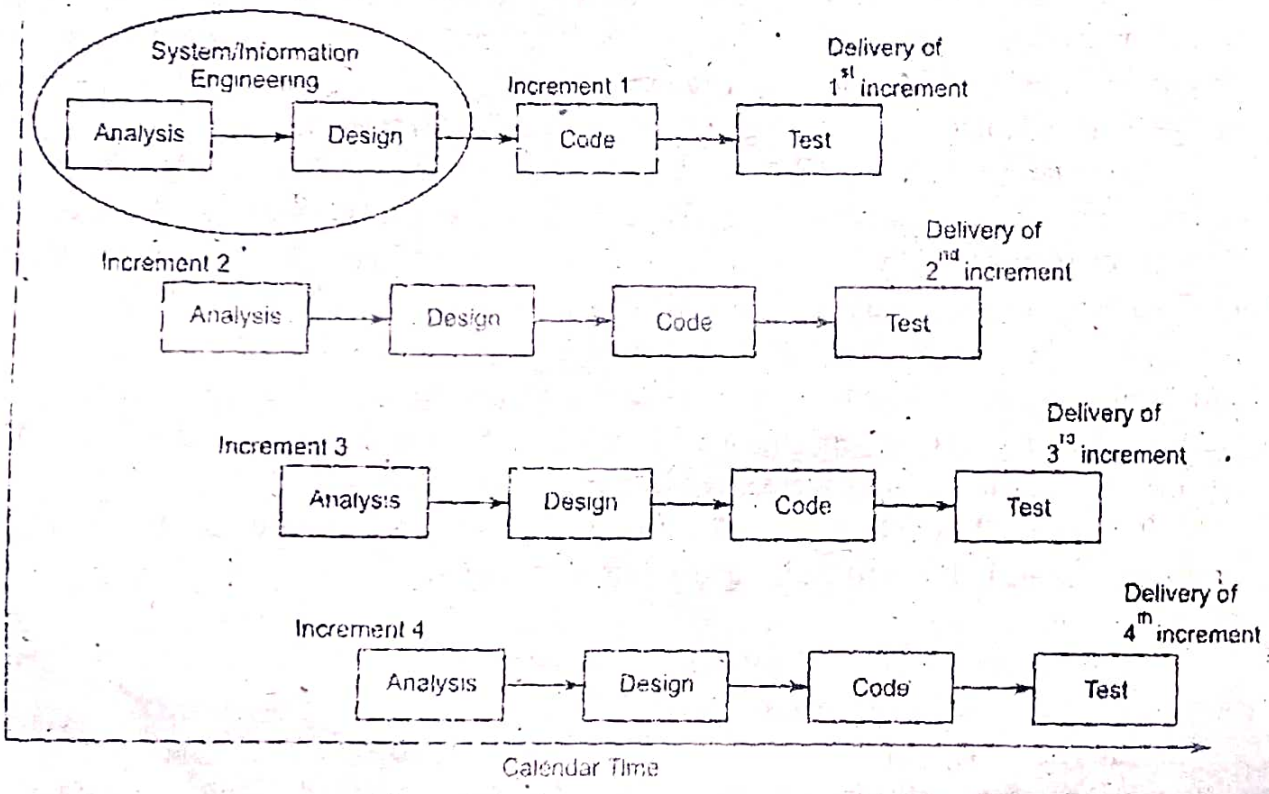


Fig : Iterative Enhancement Model

When an iterative enhancement model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, other unknown) remain undelivered. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

Advantages of Iterative Enhancement Model :

(i) The feedbacks from early increments improve the later stages.

(ii) The possibility of changes in requirement is reduced because of the shorter time span between the design of a component and its delivery.

(iii) Users get benefits earlier than with a conventional approach.

(iv) Early delivery of some useful components improves cash flow, because you get some return on investment early on.

(v) Smaller sub-projects are easier to control and manage.

(vi) 'Gold-plating', that is the requesting of features that are unnecessary and not in fact used, is less as users will know that they get more than one bite of the cherry if a feature is not in the current increment then it can be included in the next.

(vii) The project can be temporarily abandoned if more urgent work crops up.

(viii) Job satisfaction is increased for developers who see their labours bearing fruit at regular, short, intervals.

Disadvantages of Iterative Enhancement Model :

(i) Software breakage, that is, later increments may require modifications to earlier increments.

(ii) Programmers may be more productive working on one large system than on a series of smaller ones.

(iii) Some problems are difficult to divide into functional units (modules), which can be incrementally developed and delivered.

(iv) Testing of modules result into overhead and increased cost.

Section - B

Q.4(a) Describe the difference between coupling and cohesion. Explain various type of cohesion. (10)

Ans. Cohesion : *"Cohesion is a natural extension of information hiding concept."* Cohesion is a measure of the relative functional strength of a module. The cohesion of a component is a measure of the closeness of the relationships between its components. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.

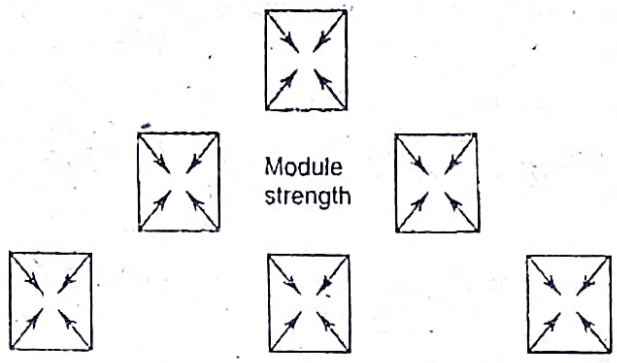


Fig (a) : Cohesion-Strength of Relation within Modules

Coupling : *"Coupling is a measure of interconnection among modules in a software structure."* The coupling between two modules indicates the degree of interdependence between them. If two modules interchange large amount of data, then they are highly interdependent. The degree of coupling between two modules depends on their interface complexity. The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

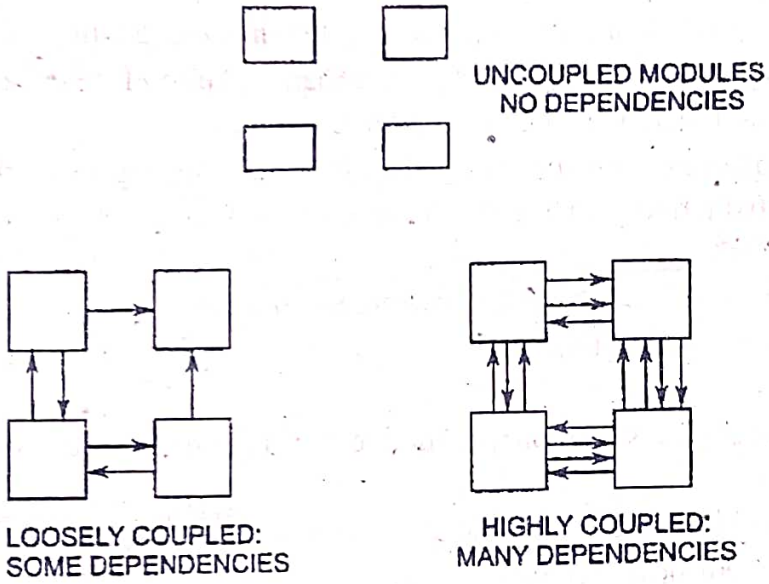


Fig.(b) : Coupling

Types of cohesion : The types of cohesion, in order of the worst to the best type, are as follows :

(i) Coincidental Cohesion : Coincidental cohesion is when parts of a module are grouped arbitrarily (at random); the parts have no significant relationship (e.g. a module of frequently used function).

(ii) Logical Cohesion : Logical cohesion is when parts of a module are grouped because they logically are categorised to do the samething, even if they are different by nature (e.g. grouping all I/O handling routines).

(iii) Temporal Cohesion : Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

(iv) Procedural Cohesion : Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

(v) 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).

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

Functional Cohesion	Best (high)
Sequential Cohesion	
Communicational Cohesion	
Procedural Cohesion	
Temporal Cohesion	
Logical Cohesion	
Coincidental Cohesion	Worst (low)

Fig (b) : The Types of Module Cohesion

Q.4(b) What is software design? Discuss the design principles in detail. (10)

Ans. Software design : Software design is the activity of specifying the nature and composition of a software system satisfying client needs and desires, subject to design constraints".

Design principles : Principle in designing a software are as follows :

(i) Abstraction : Abstraction (separates concepts from instantiations). It allows designers to focus on solving a problem without being concerned about irrelevant lower level details.

– Procedural abstraction - named sequence of events.
– Data abstraction - named collection of data objects.

(ii) Refinement : It's the process of elaboration where the designer provides successively more detail for each design component.

– Decompose design decisioin top – down to elementary level.
– Isolate design aspects that are not independent.
– Add details incrementally each step.
– Postpone decisions relating to detailed representations.
– Continually demonstrate that each refinement step is a correct expansion of previous steps.

(iii) Modularity : A module is a named entity that :

– Contains instructions, processing logic, and data structures.
– Can be separately compiled and stored in a library.
– Can be included in a program.
– Module segments can be used by invoking a name and some parameters.
– Modules can use other modules.

Modularity is the degree to which software can be understood by examining its components independently of one another.

(iv) Software Architecture : Overall structure of the software components and the ways in which that structure provides conceptual integrity for a system.

Software architecture is a sketchy map of the system. Software architecture describes the coarse grain components (usually descibes the computation) of the system. The connectors between these components describe the communication, which are explicit and pictured in a

relatively detailed way. In the implementation phase, the coarse components are refined into "actual components", e.g, classes, and Objects. In the object-oriented field, the connectors are usually implemented as interfaces, Control hierarchy or program structure.

(v) Structural Planning : In *horizontal paritioning* the control modules are used to co-ordinate communication between and execution of the functions. Partitioning this way provides the following benefits :

(i) Results in software that is easier to test and maintain.

(ii) Results in fewer propagation side-effects.

(iii) Results in software that is easier to extend.

In *vertical partitioning* the control (decision making) modules are located at the top, and work is distributed top-down in the program structure. That is, top level functions perform control functions and do little actual processing, while modules that are low in the structure perform all input, computation and output tasks. As changes to programs usually revolves around one of these three tasks there is less likelihood that changes made to the lower modules will propagate (upwards) making this partitioning strategy more maintainable.

(vi) Data Structure : It's the representation of the logical relationship among individual data elements. (requires at least as such attention as algorithm design).

Software Procedure : The precise specification of processing :

– Event sequences.
– Decision points.
– Repetitive operations.
– Data organization/structure.

(vii) Information Hiding : The principle of information hiding is the hiding of design decision in a computer program that are most likely to change, thus protecting other parts of the program from change if the design decision is changed. Information (data and procedure) contained within a module is inaccessable to modules that have no need for such information. It should be used to guide architectural design, interface designs, and modularization. Modules hide difficult or changeable design decisions.

Q.5(a) What is software requirement specification ? How it is prepared? Explain. **(10)**

Ans. Software requirement specification document is a technical specification of requirements for the software product. The goal of software requirement definition is to completely and consistently specify the technical requirements for the software product in a concise and unambiguous manner. It is based on the system definition document.

The format of the specification document is given below :

1. Product Overview and Summary
2. Development, Operating and Maintenance Environments
3. External Interfaces and Data Flow
4. Functional Requirements
5. Performance Requirements
6. Exception Handling
7. Early Subsets and Implementation Priorities
8. Foreseeable Modification and Enhancements

9. Acceptance Criteria
10. Design hints and Guidelines
11. Cross Reference Index
12. Glossary of Terms.

Sections 1 and 2 present an overview of product features and summarizes the processing environments for development, operation and maintenance of the product.

Section 3 specifies the externally observable characteristics of the software product. It includes user displays and report formats, a summary of user commands and report options, data flow diagrams and a data dictionary. High level data flow diagrams and a data dictionary are derived in this section.

Section 4 specifies the functional requrements for the software product. Functional requirements are expressed in relational and state-oriented notaions that specify relationship among inputs, actions and outputs.

Performance characteristics such as response time for various activities, processing time for various process, throughput, primary and secondary memory constraints, required telecommunication bandwidth and special issues like security constraints, reliability requirements etc. are specified in section 5.

Exception handling, including actions to be taken and the messages to be displayed in response to events are described in section 6. Categories of exceptions include temporary resource failures, out of range-input data, capacity overload etc.

The early subsets and implementation priorities for the system under development are discussed in section 7. Software products are developed as a series of successive versions and the initial version may be the prototype demonstrating basic functions and capabilities. Each successive version can incorporate the capabilities of previous versions and provide additional processing functions.

Foreseeable modifications and enchancements may be incorporated in section 8.

The software product acceptance criteria are specified in section 9. Acceptance criteria specify functional and performance tests that must be performed, the standards to be applied to source code, internal documentation and external documents such as design specifications, test plans, user manual, installation and maintenance procedures.

Section 10 contains design hints and guidelines. It is concerned with functional and performance aspects of the software product.

Section 11 relates to the sources of information used in deriving the requirements. Knowing the sources of specific requirements permits verification and re-examination of requirements, constraints and assumptions.

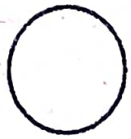
Section 12 provides the definition of terms that may be unfamiliar to the customer and the product developer.

Q.5(b) Describe the basic component of a DFD. Draw a DFD describe inventory management system. (10)

Ans. Data Flow Diagram is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output.

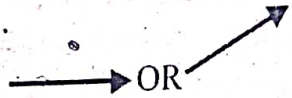
There are different types of symbols used to construct DFDs. The meaning of each symbol is explained below:

(1) Function symbol : A function is represented using a circle. This symbol is called a process or a bubble or performs some processing of input data.

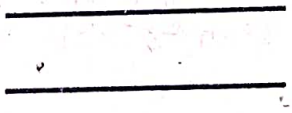


(2) External entity : A square defines a source of destination of system data. External entities represent any entity that supplies or receives information from the system but is not a part of the system.

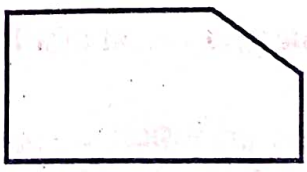
(3)Data flow symbol : A directed arc or arrow is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.



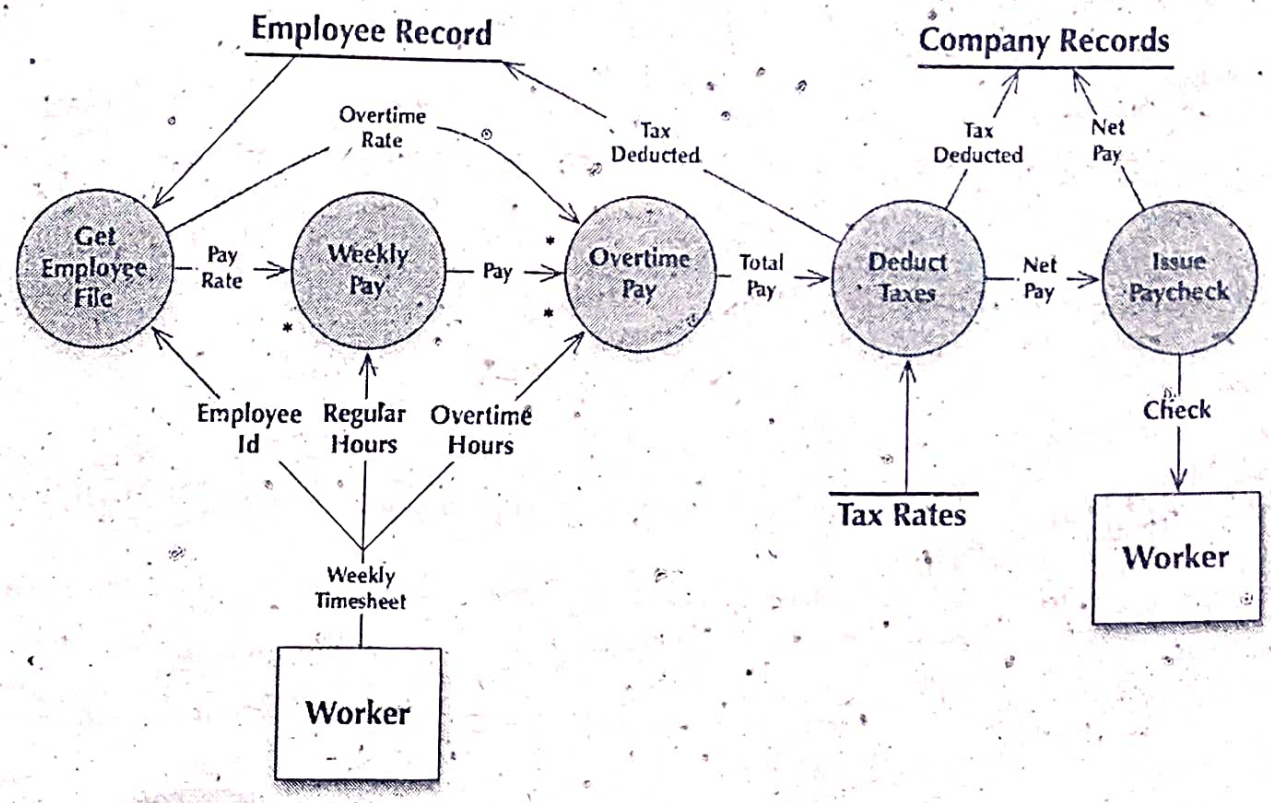
(4) Data store symbol : A data store symbol is represented using two parallel lines. A logical file can represent either a data store symbol, which can represent either a data structure, or a physical file on disk. Each data store is connected to a process by means of a data flow symbol. The direction of the data flow arrow shows whether data is being read from or written into a data store.



(5) Output symbol : It is used to represent data acquisition and production during human computer-interaction.



Draw a DFD describe inventory management system :



Section - C

Q.6(a) Design various test cases to find out the root of a quadratic equation using various method of functional testing. **(10)**

Ans. Refer Q.6.(a) of paper May 2013.

Q.6(b) Differentiate between a database and data warehouse. What is the utility of data warehouses in current scenario. **(10)**

Ans. Some differences between a database and a data warehouse:

- A database is used for Online Transactional Processing (OLTP) but can be used for other purposes such as Data Warehousing.
- A data warehouse is used for Online Analytical Processing (OLAP). This reads the historical data for the Users for business decisions.
- In a database the tables and joins are complex since they are normalized for RDMS. This reduces redundant data and saves storage space.
- In data warehouse, the tables and joins are simple since they are de-normalized. This is done to reduce the response time for analytical queries.
- Relational modeling techniques are used for RDMS database design, whereas modeling techniques are used for the Data Warehouse design.
- A database is optimized for write operation, while a data warehouse is optimized for read operations.
- In a database, the performance is low for analysis queries, while in a data warehouse, there is high performance for analytical queries.
- A data warehouse is a step ahead of a database. It includes a database in its structure.

A data warehouse is a database used to store data. It is a central repository of data in which data from various sources is stored. A data warehouse is also known as an enterprise data warehouse.

The data warehouse is then used for reporting and data analysis. It can be used for creating trending reports for senior management reporting such as annual and quarterly comparisons.

The purpose of a data warehouse is to provide flexible access to the data to the user. Data warehousing generally refers to the combination of many different databases across an entire enterprise. Data warehouses store current as well as historical data, so that all of the relevant data may be used for analysis. The analysis helps to find and show relationships among the data, to extract meaning from the data.

Q.7(a) Describe reverse engineering and Re-engineering and differntiate between the two. **(10)**

Ans. Reverse engineering : Reverse engineering is the process of analyzing software with the objective of recovering its design and specification.

OR

Reverse engineering is the process followed in order to find difficult ,unknown and hidden information about a software system.

Purpose of reverse engineering : The main purpose of engineering is to recover information from the existing code or any other intermediate documents, an activity that requires program understanding at any level may fall wihin the scope of reverse engineering. The objective of reverse engineering is to derive the design or specification of a system from its source code.

Reverse engineering process : The reverse engineering process is illustrated in fig.(a). The process starts with an analysis phase. During this phase, the system is analyzed using automated tools to discover its structure. In itself, this is not enough to re-create the system design. Engineers then work with the system source code and its structural model. They add information to this, which they have collected by understanding the system. This information is maintained as a directed graph that is linked to the program source code.

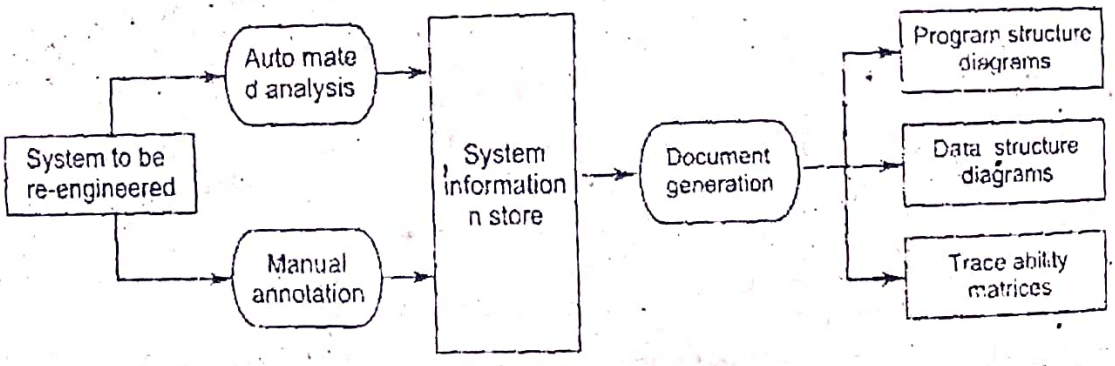


Fig.(a) : The reverse engineering process

Information store browsers are used to compare the graph structure and the code and to annotate the graph with extra information. Documents of various types such as program and data structure diagrams and trace ability metrices can be generated from the directed graph. Trace

ability metrices show where entities in the system are defined and referenced. The process of document generation is an iterative one as the design information is used to further refine the information held in the system repository.

Re-engineering : *Re-engineering means to re-implementing systems to make them more maintainable.*

In re-engineering the functionality and system architecture remains the same but it includes re-documenting, organizing and restricting, modifying and updating the system. It is a solution to the problems of system evolution. In other words,

Re-engineering essentially means having a re-look on an entity, such as process, task, design, approach or strategy using engineering principles to bring in radical and dramatic improvement.

The re-engineering approch attacks five parameters, namely : management philosophy, pride, policy, procedures and practises to bring in radical improvement impacting cost, quality, service and speed. When re-engineering principles are applied to business process then it is called Business Process Re-engineering(BRP). The objective of re-engineering is to produce, new, more maintainable system.

Re-engineering Process : Fig.(b) illustrates a possible re-engineering process. The input to the process is a legacy program and the output is a structured, modularized version of the same program. At the same time as program re-engineering, the data for the system may also be re-engineered.

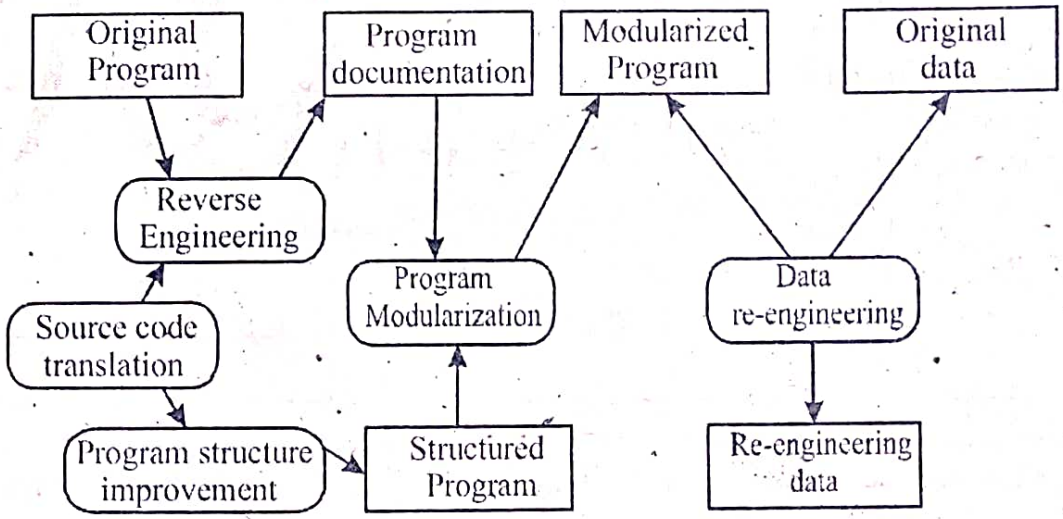


Fig.(b) : Re-engineering Process

Re-engineering process includes the following activities :

– *Source code translation :* In it the programming language of an old program is converted into the modern version of the same language or to a new language.

– *Reverse engineering :* In it the program is analyzed and important and useful information are extracted from it which helps to document its functionality.

– *Program structure improvement :* In it control structure of the program is analyzed and modified to make it easier to read and understand.

– *Program modularization :* In it redundancy of any part is removed and related parts are grouped together.

– *Data Re-engineering* : In it the data processed by the program is change to reflect program changes.

Q.7(b) Differentiate between : **(10)**

(i) Alpha and Beta testing

(ii) Verification and Validation

Ans. (i) Alpha and Beta testing :

Alpha Testing : It is is conducted at the Developer sight by end users. It is conducted in a controlled environment.

Alpha Testing is performed by a team member of software developer in the organization, to test each and every condition depending upon customer requirements and they will satisfy themselves, that software must not failure toward the client side.

Alpha testing, process continues until the system developer and the client agrees that delivered system is an acceptable implementation of the system requirement.

Beta Testing : It is conducted at the end users site for specific period of time. Unlike alpha testing, the developer is generally not present here, during testing time. Therefore, beta test is a "live" application of the software in an environment, i.e. not controlled by developer. The end users record all problems (real or imagined) that are encountered during beta tesing and reports these to developer at regular interval. As a result of problem reported during beta tests, software engineers make and then prepare for release of software product to entire customer base.

Any kind of problem arise during testing, take action to recover them an ensure that these kind of problem are not arise in future.

It involves delivering a syustem to a number of potential customer, who agree to use the system. They report problems to the system developer.

(ii) Verification and Validation

Difference between verification and validation are as follows :

Verification	Validation
(i) Verification ensures that the product is designed to deliver all functionality to the customer.	(i) Validation ensures that functionality is the intended behaviour of the product.
(ii) It involves reviews and meetings to evaluate documents, plans, code, requirements and specifications.	(ii) It involves actual testing and takes place after verification is completed.
(iii) It evaluates documents, plans, code, requirements and specifications.	(iii) It evaluates the product itself.
(iv) Inputs are checklists, issues lists, inspection meetings, reviews and meetings.	(iv) It is the actual testing of the actual product.
(v) Output is a nearly perfect set of documents, plans specifications and requirements document.	(v) Output is a nearly perfect product.

Section - D

Q.8(a) What is software quality ? Discuss the software quality attributes. (10)

Ans. Software Qualtity : Software quality is the "Conformance to explicit stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software".

Major **software quality attributes** include the following :

(i) Correctness : Correctness is a mathematical property that establishes the equivalence between the software and its specification. The correctness of a software system refers to :

– Agreement of *program code with specifications*

– Independence of the actual application of the software system.

(ii) Reliabilty : The specialised literature on software reliability defines reliability in terms of statistical behaviour the probability that the software will operate a expected over a specified time interval.

Reliability of a software system derives from

– Correctness, and

– Availability.

(iii) User Friendliness : A software system is user friendly if its human users find it easy to use. This definition reflects the subjective nature of user friendless.

Use friendliness of any software system is characterised by the following :

– Learnability

– Robustness

Learnability : Learnability of a software system depends on :

– The design of user interfaces

– The clarity and the simplicity of the user instructions (or manuals).

Robustness : A software system is robust if the consequences of an error in its operation, in the input, or in the hardware, in relation to a given application, are inversely proportional to the probability of the occurrence of this error in the given application.

(iv) Verifiability : Verifiability is usually an internal quality of software system, although it sometimes becomes an external quality also. A software system is verifiable if its properties can be verified easily. Verification can be performed either by formal analysis methods or through testing.

(v) Reusability : Reusability is analogous to evoluability. In product evolution, we modify a product to build a new version of the same product; in product reuse, we use it with minor changes to build another product.

(vi) Maintainability : Maintainability of a software system may be defined as the suitabilty for debugging, and ease for modifieation and extension of functionality.

The maintainability of a software system depends on its :

– Readability

– Extensibility

– Testability

Readability : Readability of a software system depends on its:

– Form of representation

– Programming style

– Consistency

– Readability of the implementation programming languages
– Structuredness of the system
– Quality of the documentation
– Tools available

Extensibility : Extensibility allows required modifications at the appropriate locations to be made without undesirable side effects. Extensibility of a software system depends on its :

– Structuredness (modularity) of the software system
– Possibilities that the implementation language provides for this purpose
– Readability (to find the appropriate location) of the code
– Availability of comprehensible program documentation

Testability : Testability of a software system is the suitability for allowing the programmer to follow program execution (run-time behaviour under given conditions) and for debugging. The testability of a software system depends on its :

– Modularity
– Structuredness

(vii) Efficiency : Efficiency of a software system is the ability of a software system to fulfil its purpose with the best possible utilization of all necessary resources such as time, storage transmission channels, peripherals, etc.

(viii) Portability : Software system portability is the ease with which a software system can be adapted to run on computers other than the one for which it was designed.

The portability of a software system depends on the following :

– Degree of hardware independence
– Implementation language
– Extent of exploitation of specialized system functions
– Hardware properties
– Structuredness : System-depoendent elements are collected in easily interchangeable program components.

Q.8(b) What do you mean by software configuration management ? Discuss.(10)

Ans. Software configuration management : Software configuration management (SCM) is one of the foundations of software engineering. It is used to track and manage the emerging product and its versions. This is to assure quality of the product during development, and operational maintenance of the product. SCM ensures that all people involved in the software process know what is being designed, developed, built, tested, and delivered.

Definitions : Different people perceive software Configuration Management differently. The following are few perceptions of several people about SCM in the form of definitions.

(i) Software Configuration Management (SCM) is the art of identifying organizing and controlling modifications to the software being built by a programming team. The goal of is to maximize productivity by minimizing mistakes.

(ii) Software Configuration Management (SCM) is the process of defining and implementing a standard configuration, that results into the primary benefits such as easier set-up and maintenance, less down time, better integration with enterprise management, and more efficient and reliable backups.

(iii) Software Configuration Management (SCM) is the process concerned with the development of procedures and standards for managing an evolving software system product.

(iv) Software Configuration Management is the ability to control and manage change in a software project.

Importance of SCM : SCM gives developers a means to coordinate the work of numerous people on a common product whether they work in close proximity or over a wide geographical area. Without a good SCM plan, projects become increasingly difficult to control and can reach the point where the project has to be discontinued because it can not be fixed. There is an impact of software lifecycle model on software configuration management.

Goals : The following are the major goals of software configuration management :

(i) Software configuration management activities are planned.

(ii) Selected software work products are identified, controlled and available.

(iii) Change to identified software work products are controlled.

(iv) Affected groups and individuals are informed of the status and content of software baseline.

Q.9(a) Explain SQA activities in details. (10)

Ans. SQA is the process of evaluating the quality of a product and enforcing adherence to software product standards and procedures. It is an umbrella activity that ensures conformance to standards and procedures throughout the SDLC of a software product. These include:

(i) Formulating a quality management plan : The quality management plan identifies the quality aspects of the software product to be developed. It helps in planning checkpoints for work products and the development process. It also tracks changes made to the development process based on the results of the checks.

(ii) Applying software engineering techniques : It helps the software designer to achieve high quality specification. The designer gathers information using techniques such as interviews. Using the information gathered, the designer prepares project estimation with the help of techniques such as SLOC, FP estimation.

(iii) Conducting formal technical reviews : Formal technical review is conducted to assess the quality and design of the prototype. It is a meeting with the technical staff to discuss the quality requirements of a software product and its design quality. It helps in detecting errors at an early phase of development.

(iv) Applying a multi-tiered testing strategy : Software testing is a critical task of SQA activity, which aims at error detection. Unit testing is the first level of testing. The subsequent levels of testing are integration and system level testing. There are various testing strategies followed by an organization.

(v) Enforcing process adherence : It emphasizes the need for process adherence during product development. The development process should also adhere to procedures defined for product development. It is a combination of product evaluation and process monitoring.

(vi) Controlling change : It combines human procedures and automated tools to provide a mechanism for change control. It ensures software quality by formalizing requests for change, evaluating the nature of change and controlling the impact of change.

(vii) Measuring impact of change : Changes need to be measured and monitored. Changes are measured using software quality metrics. Software quality metrics helps in estimating the cost and resource requirements. It is essential to measure quality and then compares it with the established standards.

(viii) Performing SQA audits : They scrutinize the software development process by comparing it with the established processes. It ensures that the proper control is maintained over the documents required during SDLC.

(ix) Keeping records and reporting : Keeping records and reporting ensure the collection and circulation of information relevant to SQA. The result of reviews, audits, testing etc. are reported and compiled for future reference.

Q.9(b) What is meant by ISO ? Discuss the relative merits of ISO 9000 certification. **(10)**

Ans. ISO : The International Organization for Standardization (ISO) is worldwide federation of national standards bodies from some 100 countries. ISO is a non-governmental organization established in 1947.

ISO 9000 is a series of standards dealing with quality management systems. The standards are published by the International Organization for Standardization. All industrialized countries are member and participate in writing the standards.

Most countries have adopted and published ISO 9000 as their own national standard. In the United States, it has been issued as Q9000 with virtually the same text as the original standard.

Merits of ISO 9000 certification to various organizations can be enumerated as under:

1. Continuous Improvement : Business ISO-9000 forces an organization to focus on "how they do business". Each procedure and work instruction must be documented and thus, becomes the springboard for continuous Improvement.

2. Eliminate Variation : Documented processes are the basis for repetition and help eliminate variation within the process. As variation is eliminated, efficiency improves. As efficiency improves, the cost of quality is reduced.

3. Higher Real and Perceived Quality :

(i) With the development of solid Corrective and Preventative measures, permanent, company-wide solutions to quality problems are found.

(ii) This results in higher quality.

4. Boost Employee Morale : Employee morale is increased as they are asked to take control of their processes and document their work processes.

5. Improved Customer Satisfaction : Customer satisfaction, and more importantly customer loyalty, grows as a company transforms from a reactive organization to a pro-active, preventative organization. It becomes a company people want to do business with.

6. Increased Employee Participation : Reduced problems resulting from increased employee participation, involvement, awareness and systematic employee training.

7. Better product and Services : Better products and services result from Continuous Improvement processes.

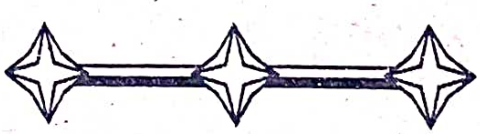
8 Greater Quality Assurance : Fosters the understanding that quality, in and of itself, is not limited to a quality department but is everyone's responsibility.

9. Improved Profit Levels : Improved profit levels result as productivity improves and rework costs are reduced.

10. Improved Communication : Improved communication both internally and externally which improves quality, efficiency, on time delivery and customer/supplier relations.

11. Reduced Cost : ISO standards results into a reduced cost of the product.

12. Competitive Edge : In order to serve to deal with the increased business complexity and for offering higher customer services, ISO 9000 standards adds as competitive edge.



PRINCIPLES OF SOFTWARES

May 2015
Paper Code: CSE-302-F

Note: ***Attempt five questions in all, selecting one question from each section. Q. No. 1 is compulsory.***

Q.1.(a) What do you mean by software project management? (5)

Ans. Software project management is the art and science of planning and leading of software projects. It is a sub-discipline of project management in which projects are planned, monitored and controlled.

We need to look at some key ideas about the planning, monitoring and control of software projects. Projects to produce software are worthwhile only if they satisfy real needs and so we will examine how we can identify the stakeholders in a project and their objectives. Identifying those objectives and checking that they are met is the basis of a successful project. This, however, cannot be done unless there is accurate information and how this is provided will be explored.

A software project has two main activity dimensions: engineering and project management. The engineering dimension deals with building the system and focuses on issues such as how to design, test, code and so on. The project management dimension deals with properly planning and controlling the engineering activities to meet project goal for cost, schedule, and quality.

Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development, and support of computer software. Software project management encompasses the knowledge, techniques, and tools necessary to manage the development of software products.

Q.1.(b) Describe software crisis and reason for it.

Ans. Software Crisis : The *software crisis* is characterized by *an inability to develop software on time, on budget and within requirements*. As a result, the delivered software systems are :

(i) Completely unsatisfactory (not as per the specification)
(ii) Extremely late.
(iii) Far over the budget.
(iv) Poorly suited for intended user of the system.
(v) Projects running over-time.
(vi) Software was of low qualtity.
(vii) Software often did not meet requirements.
(viii) Projects were unmanageable & code difficult to maintain.

Various **reason of crisis** are as follows :

(i) Lack of communication between software developers and users.
(ii) Increase in cost of software compared to hardware.
(iii) Increase in the size of software.
(iv) Increased complexity of the porblem area.
(v) Project management problem.

(vi) Lack of understanding of the problem and its enviornment and
(vii) High optimistic estimates regarding software development time and cost
(viii) Duplication of efforts due to absent of automation in most of the software development activites.

Q.1.(c) Explain the significance of software re-engineering. **(5)**

Ans. The principles of re-engineering when applied to software-development processes are called software re-engineering. It affects positively software cost, quality, service to the customer, and speed of delivery. Software, whether a product or system, deals with business processes making them faster, smarter, and automatic in response to delivery and execution.

Following are the some significance of software re-engineering :

(1) Redefining software scope and goals.
(2) Redefining RDD and SRS by way of additions, deletions, and extensions of functions and features.
(3) Redesigning the application design and architecture using new technology, upgrades, and platforms, interfacing to new technologies to make the process faster, smarter, and automatic.
(4) Resorting to data restructuring, improving database design, code restructuring to make the size smaller and more efficient in operations.
(5) Rewriting the documentation to make it more user friendly.

Q.1.(d) What do you mean by verification and validation? **(5)**

Ans. Verification and Validation (V & V) : An important objective of software testing is verification and validation (V & V). Testing can serve as metrics. It is heavily used as a tool in the V & V process. Testers can make claims based on interpretations of the testing results, which either the product works under certain situations, or it does not work.

Software quality cannot be tested directly but the related factors to make quality visible can be tested. Quality has three sets of factors:

1. Functionality.
2. Engineering. and
3. Adoptability.

These three sets of factors can be thought of as dimensions in the software quality space. Each dimension may be broken down into its component factors and considerations at successively lower levels of detail. Table illustrates some of the most frequently cited quality considerations suggested by Hetzel in 1988.

Table Typical Software Quality Factors

Functionality (exterior quality)	Engineering (interior quality)	Adaptability (future quality)
Correctness	Efficiency	Flexibility
Reliability	Testability	Reusability
Usability	Documentation	Maintainability
Integrity	Stucture	

Good testing provides measures for all relevant factors. The importance of any particular factor varies from application to application. Any system where human lives are at stake must place extreme emphasis on reliability and integrity. In the typical business system usability and

maintainability are the key factors, while for a one-time scientific program neither may be significant. Our testing, to be fully effective, must be geared to measuring each relevant factor and thus forcing quality to become tangible and visible.

Tests with the purpose of validating the product works are named clean tests, or positive tests. The drawbacks are that it can only validate that the software works for the specified test cases. A finite number of tests cannot validate that the software works for all situations. On the contrary, only one failed test is sufficient enough to show that the software does not work.

Dirty tests, or negative tests, refer to the tests aiming at breaking the software, or showing that it does not work. A piece of software must have sufficient exception handling capabilities to survive a significant level of dirty tests.

A testable design is a design that can be easily validated, falsified and maintained. Because testing is a rigorous effort and requires significant time and cost, design for testability is also an important design rule for software development.

Section-A

Q.2.(a) Define the term "Software engineering". Explain the major differences between software engineering and other traditional engineering disciplines. (10)

Ans. Few important definitions of software engineering are:

IEEE Comprehensive Definition : Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software.

According to Barry Boehm : Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation.

Differences between software engineering and traditional engineering disciplines:

Issue	Software Engineering	Traditional Engineering
Foundations	Based on computer science, information science, and discrete math.	Based on science, mathematics, and empirical knowledge.
Cost	Compilers and computers are cheap, so software engineering and consulting are often more than half of the cost of a project. Minor software engineering cost-overruns can adversely affect the total project cost.	In some projects, construction and manufacturing costs can be high, so engineering may only be 15% of the cost of a project. Major engineering cost overruns may not affect the total project cost.
Replication	Replication (copying CDs or downloading files) is trivial. Most development effort goes into building new (unproven) or changing old designs and adding features.	Radically new or one-of-a-kind systems can require significiant development effort to create a new design or change an existing design. Other kinds of systems may require less development effort, but more attention to issues such as manufacturability.

Innovation	Software engineers often apply new and untested elements in software projects.	Engineers generally try to apply known and tested principles, and limit the use of untested innovations to only those necessary to create a product that meets its requirements.
Duration	Software engineers emphasize projects that will live for years or decades.	Some engineers solve long-ranged problems (bridges and dams) that endure for centuries.
Management Status	Few software engineers manage anyone.	Engineers in some disciplines, such as civil engineering, manage construction, manufacturing, or maintenance crews.
Blame	Software engineers must blame themselves for project problems.	Engineers in some fields can often blame construction, manufacturing, or maintenance crews for project problems.
Practitioners in U.S. Age	611,900 software engineers. Software engineering is about 50 years old.	1,157,020 total non-software engineers. Engineering as a whole is thousands of years old.
Title Regulations	Software engineers are typically self-appointed. A computer science degree is common but not at all a formal requirement.	In many jurisdictions it is illegal to call yourself an engineer without specific formal education and / or accreditation by governmental or engineering association bodies.
Analysis Methodology	Methods for formally verifying correctness are developed in computer science but they are rarely used by software engineers. The issue remains controversial.	Some engineering disciplines are based on a closed system theory and can in theory prove formal correctness of a design. In practice, a lack of computing power or input data can make such proofs of correctness intractable, leading many engineers to use a pragmatic mix of analytical approximations and empirical test data to ensure that a product will meet its requirements.
Synthesis Methodology	SE struggles to synthesize (build to order) a result according to requirements.	Engineers have nominally refined synthesis techniques over the ages to provide exactly this. However, this has not prevented some notable engineering failures, such as the collapse of the Tacoma Narrows Bridge, the sinking of the Titanic, and the Pentium FDIV bug. In addition, new technologies inevitably result in new challenges that cannot be met using existing techniques.

Research during	Software engineering is often busy with researching the unknown (e.g., to derive an algorithm) right in the middle of a project.	Traditional engineering nominally separates these activities. A project is supposed to apply research result in known or new clever ways to build the desired result. However, ground-breaking engineering projects such as Project Apollo often include a lot of research into the unknown.
Codified	Software engineering has just recently started to codify and teach best practice in the form of design patterns.	Some engineering disciplines have thousands of years of best practice experience handed over from generation to generation via a field's literature, standards, rules and regulations. Newer disciplines such as electronic engineering and computer engineering have codified their own best practices as they have developed.

Q.2.(b) Compare iterative enhancement model and evolutionary process model. **(10)**

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 [BASI76].

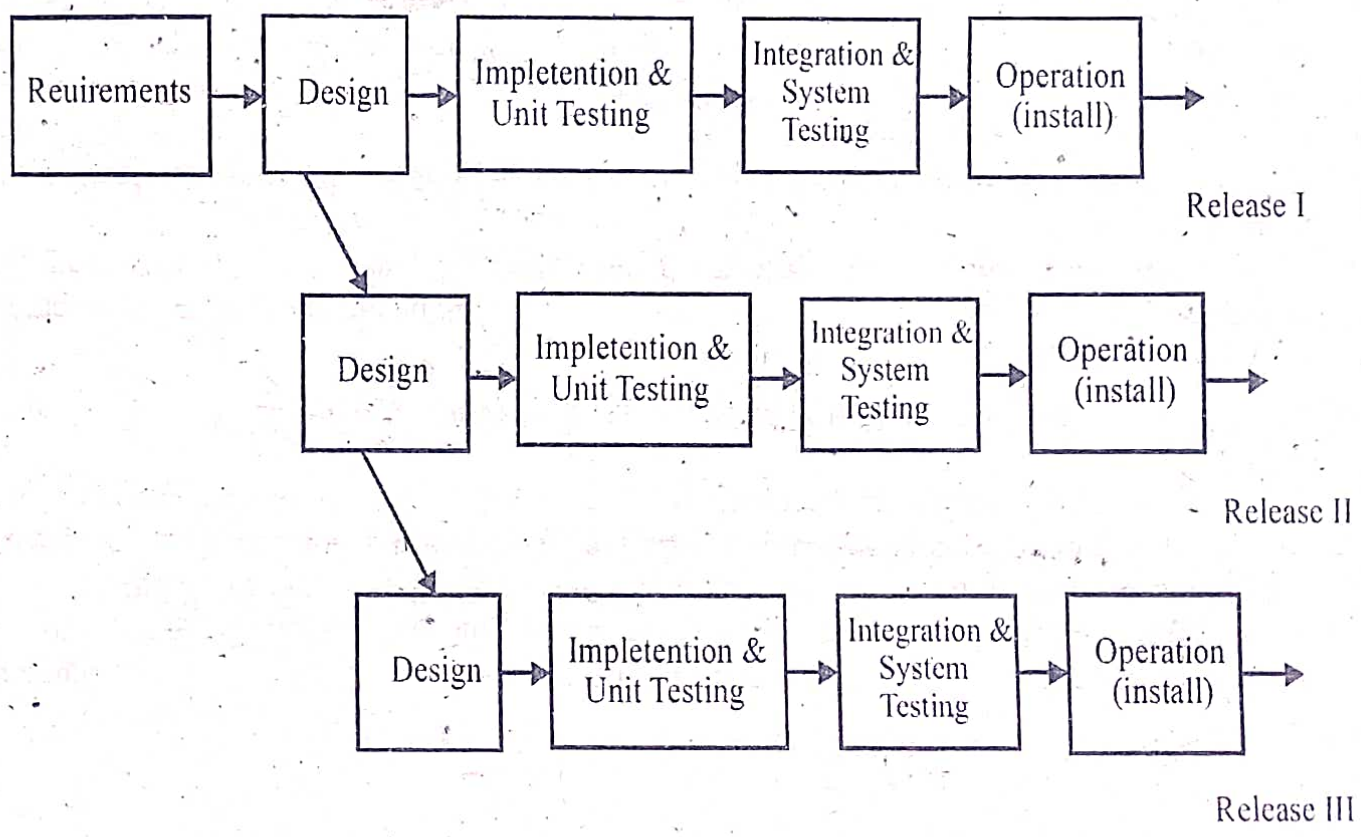


Fig. : Iterative enhancement model

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

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

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.

For example, in a simple database application, one cycle might implement the graphical user interface (GUI); another file manipulation; another queries; and another updates. All four cycles must complete before there is working product available. GUI allows the users to interact with the system; file manipulation allows data to be saved and retrieved; queries allow users to get data out of the system; and updates allow users to put data into the system. With any one of those parts missing, the system would be unusable.

In contrast, an iterative enhancement model would start by developing a very simplistic, but usable database. On the completion of each cycle, the system would become more sophisticated. It, would, however, provide all the critical functionality by the end of the first cycle. Evolutionary development and iterative enhancement are somewhat interchangeable. Evolutionary development should be used when it is not necessary to provide a minimal version of the system quickly.

This model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning.

Q.3. What do you mean by COCOMO? Explain all the levels of COCOMO model. (20)

Ans. COCOMO model : COCOMO stands for *Constructive Cost Model*. It was introduced by Barry Boehm in 1981. It is perhaps the best known and most throughly documented of all software cost estimation models. It provides the following three levels of models :

(i) *Basic COCOMO model :* The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions :

$$\text{Effort} = a_1 \times (KLOC)^{a_2} \quad PM$$

$$T_{dev} = b_1 \times (Effort)^{b_2} \quad Months$$

Where, *KLOC* is the estimated size of the software product expressed in kilo lines, Code a_1, a_2, b_1, b_2 are constants of software products.

T_{dev} is the estimated time to develop the software, expressed in months.

Effort is the total effort required to develop the software product, expressed in person-month (PM).

Person months curve is shown in fig.(1).

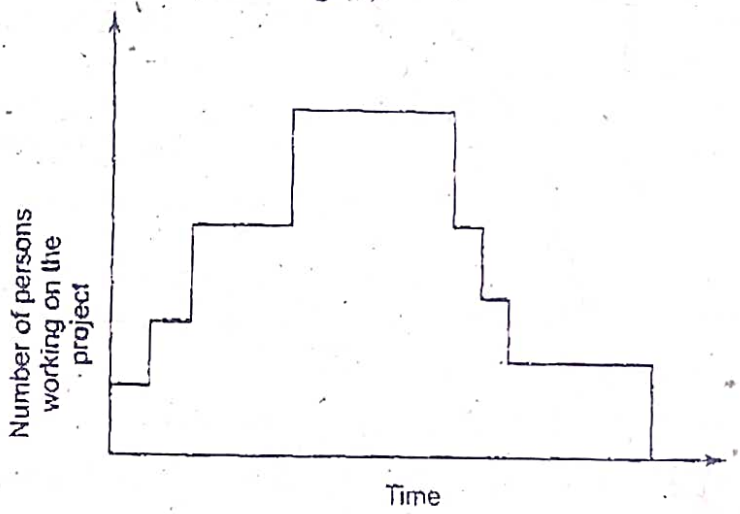


Fig.(1) : Person-month curve

Estimation of Development effort :

Organic: Effort = $2.4(KLOC)^{1.05}$ PM

Semidetached : Effort = $3.0(KLOC)^{1.12}$ PM

Embedded : Effort = $3.6(KLOC)^{1.20}$ PM

Fig.2(a) shows a plot of estimated effort versus size for various product sizes.

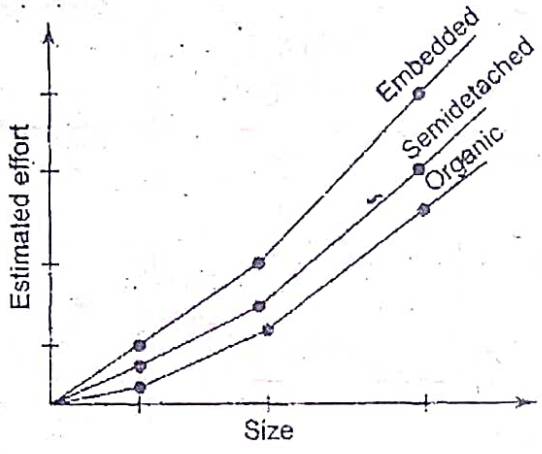


Fig.2(a) : Effort versus size

From fig.2(a), we can observe that the effort is almost linearly proportional to the size of the software product.

Estimation of Development time :

Organic: $T_{dev} = 2.5(Effort)^{0.38}$ Months

Semidetached: $T_{dev} = 2.5(Effort)^{0.35}$ Months

Embedded: $T_{dev} = 2.5(Effort)^{0.32}$ Months

Fig.2(b) shows the plot is development time versus product size.

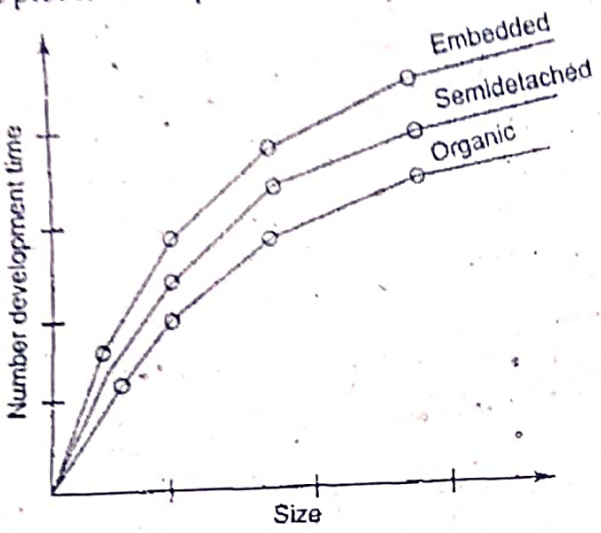


Fig.2(b) : Development Time versus size

From fig.2(b), we can observe that the development time is sub linear function of the size of the product.

(ii) ***Intermediate COCOMO Model :*** The intermediate COCOMO model recognizes this fact and refines the intial estimate obtained through the BASIC COCOMO expression by using a set of 15 cost drivers (multipliers) based on various attributes of software development.

Table : COCOMO Intermediate Cost drivers

Driver type	Code	Cost Driver
Product attributes	RELY	Required software reliability
	DATA	Database size
	CPLX	Product complexity
Computer attributes	TIME	Execution time constraints
	STOR	Main storage constraints
	VIRT	Virtual machine volatility-degree to which the operating system changes
	TURN	Computer turn around time
Personnel attributes	ACAP	Analyst capability
	AEXP	Application experience
	PCAP	Programmer capability
	VEXP	Virtual machine (i.e.,operation system) experience
	LEXP	Programming language experience
Project attributes	MODP	Use of modern programming practices
	TOOL	Use of software tools
	SCED	Required development schedule.

(iii) *Complete COCOMO Model :* The short-comings of both basic and intermediate COCOMO models are that they :

– Consider a software product as a single homogeneous entity.

– However, most large systems are made up of several smaller sub-systems. Some sub-systems may be considered as organic type, some embedded, etc. For some the reliability requirements may be high and so on.

– Cost of each sub-system is estimated separately.

– Costs of the sub-systems are added to obtain total-cost.

– Reduces the margin or error in the final estimate.

A Large amount of work has been done by Boehm to capture all significant aspects of a software development. It offers a means for processing all the project characteristics to construct a software estimate.

Section-B

Q.4.(a) Define coupling and cohesion. Explain their types. **(10)**

Ans. Cohesion : *"Cohesion is a natural extension of information hiding concept."* Cohesion is a measure of the relative functional strength of a module. The cohesion of a component is a measure of the closeness of the relatioinships between its components. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.

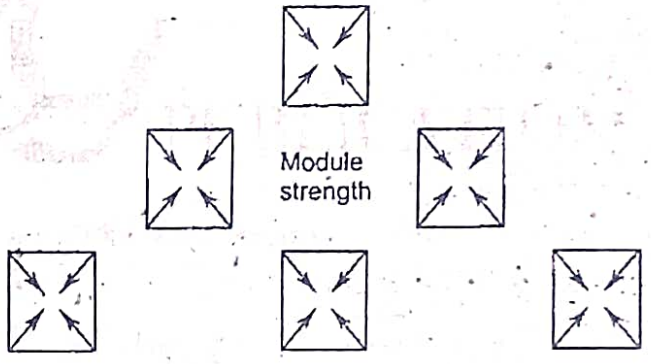


Fig (a) : Cohesion-Strength of Relation within Modules

Coupling : *"Coupling is a measure of interconnection among modules in a software structure."* The coupling between two modules indicates the degree of interdependence between them. If two modules interchange large amount of data, then they are highly interdependent. The degree of coupling between two modules depends on their interface complexity. The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

Types of cohesion : The types of cohesion, in order of the worst to the best type, are as follows :

(i) Coincidental Cohesion : Coincidental cohesion is when parts of a module are grouped arbitrarily (at random); the parts have no significant relationship (e.g. a module of frequently used function).

(ii) Logical Cohesion : Logical cohesion is when parts of a module are grouped because they logically are categorised to do the samething, even if they are different by nature (e.g. grouping all I/O handling routines).

(iii) Temporal Cohesion : Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

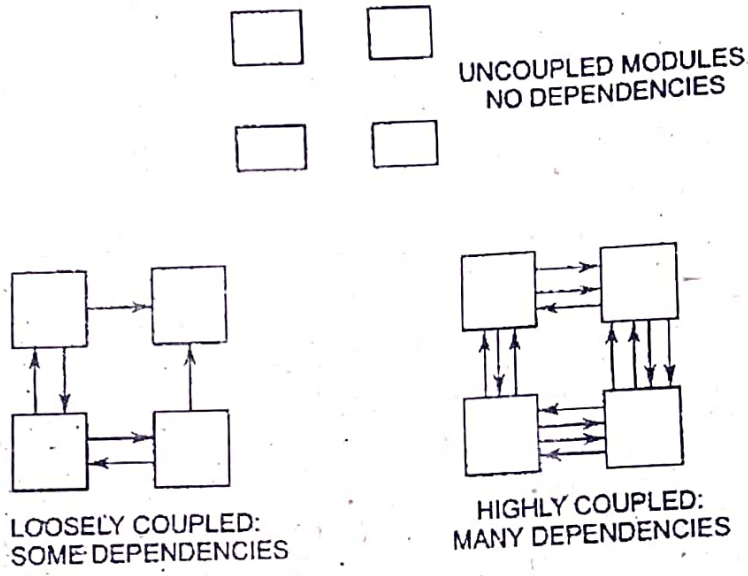


Fig.(b) : Coupling

(iv) Procedural Cohesion : Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

(v) 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).

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

Functional Cohesion	Best (high)
Sequential Cohesion	
Communicational Cohesion	
Procedural Cohesion	
Temporal Cohesion	
Logical Cohesion	
Coincidental Cohesion	Worst (low)

Fig (b) : The Types of Module Cohesion

Best Cohesion: Functional Cohesion

Worst Cohesion: Coincidental Cohesion

Types of Coupling : Different types of Coupling are as follows :

1. Content Coupling: Content coupling is when one module modifies or relies on the internal workings of another module. Therefore changing the way the second module produces data will lead to changing the dependent module.

2. Common Coupling : Common coupling is when two modules share the same global data. Changing the shared resource implies changing all the modules using it.

3. External coupling : It occurs when two modules share an externally imposed data format, communication protocol, or device interface.

4. Control coupling : Control coupling is one module controlling the flow of another, by passing it information on what to do.

5. Data coupling : Data coupling is when modules share data through.

6. Message Coupling : This is the loosest type of coupling. It can be achieved by state decentralization and component communication is done via parameters or message passing.

Best Coupling: Content Coupling

Worst Coupling: Message Coupling

Q.4.(b) List advantages of software requirement specification. Describe the desirable characteristics of a good software requirement specification. (10)

Ans. To the customers, suppliers, and other individuals, a good software requirement specification (SRS) should provide several specific benefits, such as the following :

1. Establish the basis for agreement between the customers and the suppliers on what the software product is to do : The complete description of the functions to be performed by the software specified in the SRS will assist the potential user to determine if the software specified meets their needs or how the software must be modified to meet their needs.

2. Reduce the development effort : The preparation of the SRS force the various concerned groups in the customer's organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.

3. Provide a basis for estimating costs and schedules : The dscription of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.

4. Provide a baseline for validation and verification : Organizations can develop their validation and verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.

5. Facilitate transfer : The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and supplies find it easier to transfer it to new customers.

6. Serve as a basis for enhancement : Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

Characteristics of a good software requirement specification (SRS): The following are the quality characteristics of a good SRS document :

Complete :

- SRS should be complete.
- SRS defines precisely all the live situations that will be encountered and the system's capability to successfully address them.

Consistent :

- SRS should be consistent.
- SRS capability functions and performance levels are compatible, and the required quality features (security, reliability, etc.) do not negate those capability functions.

Accurate :

- SRS precisely defines the system's capability in a real-world environment, as well as how it interfaces and interacts with it.
- This aspect of requirements is a significant problem area for many SRSs.

Modifiable :

- The logical, hierarchical structure of the SRS should facilitate any necessary modifications and that too with ia greater ease.

Ranked :

- Individual requirements of an SRS are hierarchically arranged according to stability, security, perceived ease/difficulty of implementation, or other parameter that helps in the design of that and subsequent documents.

Testable :

- SRS must be stated in such a manner that unambiguous assessment criteria can be derived from the SRS itself.

Traceable :

- Each requirement in SRS must be uniquely identified to a source (e.g. use case, government requirement, industry standard, etc:)

Unambiguous :

- SRS must contain requirements statements that can be interpreted in one way only i.e., it shoud be unambiguous.
- This is another area that creates significant problems for SRS development because of the use of natural language.

Valid :

- A valid SRS is one in which all parties and project participants can understand, analyze, accept, or approve it.
- This is one of the main reasons SRSs are written using natural language.

Verifiable :

- A verifiable SRS is consistent form one level of abstraction to another.
- Most attributes of a specification are subjective and a conclusive assessment of quality requires a technical review by domain experts.
- Using indicators of strength and weakness provide some evidence that preferred attributes are or are not present.

(20)

Q.5. Write notes on :

(a) System design

(b) E-R model

Ans. (a) Software design : Software design is the activity of specifying the nature and composition of a software system satisfying client needs and desires, subject to design constraints".

Design principles : Principle in designing a software are as follows :

(i) Abstraction : Abstraction (separtes concepts from instantiations). It allows

designers to focus on solving a problem without being concerned about irrelevant lower level details.

– Procedural abstraction - named sequence of events.
– Data abstraction - named collection of data objects.

(ii) Refinement : It's the process of elaboration where the designer provides successively more detail for each design component.

– Decompose design decisioin top – down to elementary level.
– Isolate design aspects that are not independent.
– Add details incrementally each step.
– Postpone decisions relating to detailed representations.
– Continually demonstrate that each refinement step is a correct expansion of previous steps.

(iii) Modularity : A module is a named entity that :

– Contains instructioins, processing logic, and data structures.
– Can be separately compiled and stored in a library.
– Can be included in a program.
– Module segments can be used by invoking a name and some parameters.
– Modules can use other modules.

Modularity is the degree to which software can be understood by examining its components independently of one another.

(iv) Software Architecture : Overall structure of the software components and the ways in which that structure provides conceptual integrity for a system.

Software architecture is a sketchy map of the system. Software architecture describes the coarse grain components (usually descibes the computation) of the system. The connectors between these components describe the communication, which are explicit and pictured in a relatively detailed way. In the implementation phase, the coarse components are refined into "actual components", e.g, classes, and Objects. In the object-oriented field, the connectors are usually implemented as interfaces, Control hierarchy or program structure.

(v) Structural Planning : In *horizontal paritioning* the control modules are used to co-ordinate communication between and execution of the functions. Partitioning this way provides the following benefits :

(i) Results in software that is easier to test and maintain.
(ii) Results in fewer propagation side-effects.
(iii) Results in software that is easier to extend.

In *vertical partitioning* the control (decision making) modules are located at the top, and work is distributed top-down in the program structure. That is, top level functions perform control functions and do little actual processing, while modules that are low in the structure perform all input, computation and output tasks. As changes to programs usually revolves around one of these three tasks there is less likelihood that changes made to the lower modules will propagate (upwards) making this partitioning strategy more maintainable.

(vi) Data Structure : It's the representation of the logical relationship among individual data elements. (requires at least as such attention as algorithm design).

Software Procedure : The precise specification of processing :

– Event sequences.

– Decision points.
– Repetitive operations.
– Data organization/structure.

(vii) Information Hiding : The principle of information hiding is the hiding of design decision in a computer program that are most likely to change, thus protecting other parts of the program from change if the design decision is changed. Information (data and procedure) contained within a module is inaccessable to modules that have no need for such information. It should be usedto guide architectural desing,s interface designs, and modularization. Modules hide difficult or changeable desing decisions.

Ans.(b) E-R model : In software engineering, an entity–relationship model (ER model) is a data model for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as a relational database.

The entity-relationship (E-R) model is a high-level data model. It is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationships among these objects. It was developed to facilitate database design by allowing specification of an enterprise schema, which represent the overall logical structure of a database.

Entity : An entity is an object that has its existence in the real world. It includes all those "things" about which data is collected. An entity may be a tangible object such as a student, a place or a part. It may also be non-tangible such as an event, a job title or a customer account. For example, if we say that a customer buys goods, it means customer and goods are entities.

Diagrammatically, entities are represented in rectangles.

CUSTOMER GOODS

Entities

An Entity Set : It is a set of entities of the same type that share the same properties, or attributes. The set of all persons who are customers at a given bank, example, can be defined as the entity set customer.

Attributes : Attributes are units that describe the characteristics or properties of entities. In a database, entities are represented by tables and attributes by columns. For example, a customer entity might have numerous attributes such as code, name and addresses. Similarly, the goods entity may have attributes like code and price. They are drawn in elliptical along with the entity rectangles.

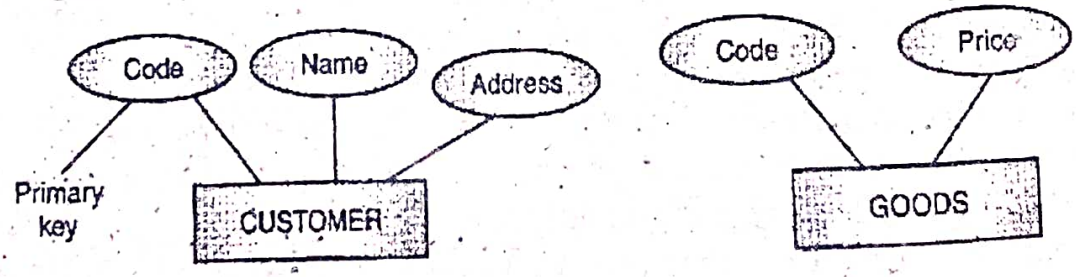


Fig. : Attributes

Domain : The domain of an attribute is the collection of all possible value, an attribute can have.

Field/Column : A column represents one related part of a table and is the smallest logical structure of storage in a database. It holds one piece of information about on item of subject.

It is generally used for a group of alphanumeric characters.

Record/Row : A record is a collection of multiple related fields that can be treated as a unit. In database terminology, each row is a record. However database experts prefer to use the word row, since record sometimes has other connotations outside of relational databases.

Table : A table is a named collection of logically related multiple records. Depending on the database software, a table can be referred to as a file.

Section-C

Q.6.(a) What are various kinds of black box testing? (10)

Ans. Black Box Testing falls in two categories:

1. Positive Functional Testing : This testing entails exercising the application's functions with valid input and verifying that the outputs are correct.

Example : Continuing with the word processing example, a positive test for the printing function might be to print a document containing both text and graphics to a Printer that is online, filled with paper and for which the correct drivers are installed.

2. Negative Functional Testing : This testing involves exercising application functionality using a combination of invalid inputs, unexpected conditions and other "out-of-bounds scenarios.

Example :

(i) Continuing the word processing example, a negative test for the printing function might be to disconnect the printier from the computer while a document is printing.

(ii) What probably should happen in this scenario is a plain-Engligh error message appears, informing the user what happend and instructing hm / her on how to remedy the problem.

(iii) What might happen instead is the word processing software simply hangs up or crashes because the "abnormal" loss of communications with the printer is not handled properly.

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

Q.6.(b) What is software maintenance. Which category consumes maximum effort ? (10)

Ans. Software Maintenance : Two formal definitions of software maintenance are given below :

IEEE, 1993 : "Software Maintenance is modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

(ISO/IEC Standard 12207 : "Software Maintenance is a set of activities performed when software undergoes modifications to code and associate documentation due to a problem or the need for improvement or adaptation."

The first definition is analogous to hardware maintenance, e.g. car servicing, where a product is checked for errors or given additional functions after it has been sold.

The second definition, on the other hand, includes software maintenance as an essential aspect of the entire life cycle of the software product, starting from its early development.

Maintenance may be classified into the four categories as follows:

- **Corrective :** reactive modification to correct discovered problems.
- **Adaptive :** modification to keep it usable in a changed or changing environment.
- **Perfective :** improve performance or maintainability.
- **Preventive :** modification to detect and correct latent faults.

(1) Corrective : Corrective maintenance means repairing processing or performance failures or making changes because of previously uncorrected problems. Corrective maintenance refers to modifications initiated by defects in the software. This type of maintenance is also called bug fixing. A defect can result from the following errors:

(a) Design Errors : Design errors occur when changes made to the software are incomplete, incorrect, wrongly communicated or the change request is misunderstood.

(b) Logic Error : Logic errors result from invalid test and conclusions, incorrect implementation of design, specifications, faulty logic flow or incomplete test data.

(c) Coding Errors : Coding errors are caused by incorrect implementation of detailed design and incorrect use of the source code logic.

(2) Adaptive Maintenance : Adaptive maintenance means changing the program function. This is done to adapt to the external environment change. For example, the current system was designed so that it calculates taxes on profits after deducting the dividend on equity shares. The government has issued orders now to include the dividend in the company profit for tax calculation. This function needs to be changed to adapt to the new system.

(3) Perfective Maintenance : Perfective maintenance means enhancing the performance or modifying the programs to respond to the user's additional or changing needs. For example, earlier data was sent from stores to headquarters on magnetic media but after stores were electronically linked via leased lines, the software was enhanced to send data via leased lines.

As maintenance is very costly and very essential, efforts have been done to reduce its costs. One way to reduce the costs is through maintenance management and software modification audit. Software are modification consists of programs rewriting, system level up gradation.

(4) Preventive Maintenance : Preventive maintenance is the process by which we prevent our system from being obsolete. Preventive maintenance involves the concept of re-engineering and reverse engineering in which an old system with an old technology is re-engineered using new technology. This maintenance prevents the system from dying out. (8)

Q.7.(a) Describe alpha and beta testing.

Ans. Alpha and Beta testing :

Alpha Testing : It is is conducted at the Developer sight by end users. It is conducted in a controlled environment.

Alpha Testing is performed by a team member of software developer in the organization, to test each and every condition depending upon customer requirements and they will satisfy themselves, that software must not failure toward the client side.

Alpha testing, process continues until the system developer and the client agrees that delivered system is an acceptable implementation of the system requirement.

Beta Testing : It is conducted at the end users site for specific period of time. Unlike alpha testing, the developer is generally not present here, during testing time. Therefore, beta test is a "live" application of the software in an environment, i.e. not controlled by developer. The end users record all problems (real or imagined) that are encountered during beta tesing and reports these to developer at regular interval. As a result of problem reported during beta tests, software engineers make and then prepare for release of software product to entire customer base.

Any kind of problem arise during testing, take action to recover them an ensure that these kind of problem are not arise in future.

It involves delivering a system to a number of potential customer, who agree to use the system. They report problems to the system developer.

Q.7.(b) What is the purpose of unit testing? How is it done? **(12)**

Ans. Unit Testing: In unit testing individual components are tested to ensure that they operate correctly. It focuses on verification effort. On the smallest unit of software design, each component is tested independently without other system components.

There are number of reasons in support of unit testing than testing the entire product.

- The size of a single module is small enough that we can locate an error fairly easily.
- The module is small enough that we can attempt to test it in some demonstrably exhaustive fashion.
- Confusing interactions of multiple errors in widely different parts of the software are eliminated.

Unit Test Procedure : The unit test environment is illustrated in Figure

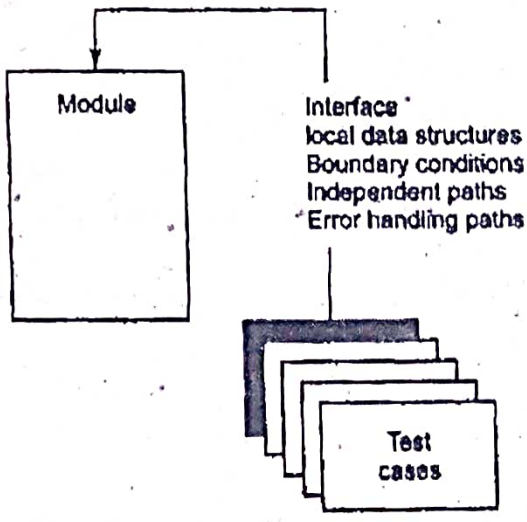


Fig. : Unit Test

In most applications a driver is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested) and prints relevant results. Stubs serve to replace modules that are subordinate (called by) to the component to be tested. A stub uses the subordinate (called by) to the component to be tested. A stub uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry and returns control to the module undergoing testing.

Drivers and stubs represent overhead. That is, both are software that must be written but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. Unfortunately, many components cannot be adequately unit tested with "simple" overhead software. In such cases, complete testing can be postponed until the integration test step (where drivers or stubs are also used).

Unit testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.

Section-D

Q.8. What is computer aided software engineering ? Explain various CASE tools. **(20)**

Ans. CASE stands for computer aided software engineering.

CASE is a tool which aids a software engineer to maintain and develop software. The workshop for software engineering is called in Integrated Project Support Environment (IPSE) and the tool set that fills the workshop is called CASE.

CASE is a computer aided software engineering technology CASE is an automated support tool for the software engineers in any software engineering process.

Software engineering mainly includes the following processes :

1. Translation of user needs into software requirements.
2. Transaction of software requirements into design specification
3. Implementation of design into code
4. Testing of the code
5. Documentation

CASE technology provides software process support by automating some process activities and by providing information about the software, which is being developed. Examples of activities, which can be automated using CASE, include:

1. The development of graphical system models as part of the requirements specification or the software design.
2. Understanding a design using a data dictionary, which holds information about the entities and relations in a design.
3. The generation of user interfaces from a graphical interface description, which is created interactively by the user.
4. Program debugging through the provision of information about an executing program.
5. The automated translation of programs from an old version of a programming language such as COBOL to a more recent version.

The use of Computer Aided Software Engneering (CASE) tool reduce the effort of development of achieving quality goals and managing change and configuration throughout the product life cycle. It also help the project manager, the software developer and other key personnel to improve their productivity in the development team.

CASE Tools : CASE Tools are software programs that are designed to assist human programmers with the complexity of the processes and the artifacts of software engineering. They constitute the laws of and the automated tools that aid in the synthesis, analysis, modelling, or documentation of software.

The schematic diagram of CASE tools is shown in fig.

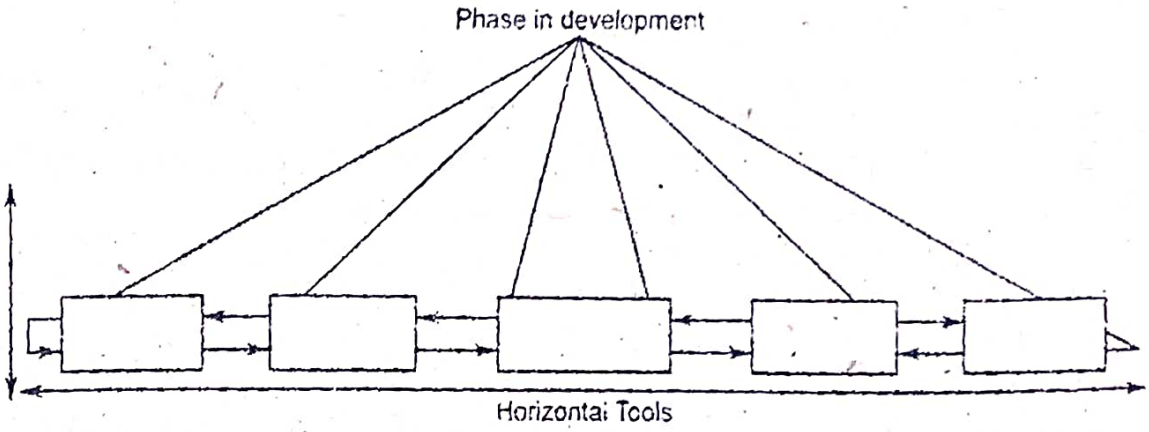


Fig. : Categories of case tools

Smith and Oman have defined CASE tools which are divided into the following two categories :

(i) Vertical CASE tools

(ii) Horizontal CASE tools

(i) Vertical CASE Tools : Vertical CASE tools provide support for certain activities within a single phase of the software life cycle.

There are two subcategories of vertical CASE tools :

– *First category :* It is the set of tools that are within one phase of life cycle. These tools are important so that development in each phase can be as quick as possible.

– *Second category :* It is a tool that is used in more than one phase, but does not support moving from one phase to the next. These tools ensure that the developer does move on the next phase as appropriate.

(ii) Horizontal CASE tools : These tools support automated transfer of information between the phases of a life cycle. These tools include project management, configuration management tools and integration services.

Various **advantages of CASE tools** are as follows :

(i) Improved productivity

(ii) Better documentation

(iii) Improved accuracy

(iv) Intangible benefits

(v) Improved quality

(vi) Reduced lifetime maintenance

(vii) Opportunity to non-programmers

(viii) Reduced cost of software

(ix) Produce high quality and consistent documents

(x) Impact on the style of a working of company

(xi) Reduce the drudgery in a software engineer's work

(xi) Increase speed of processing

(xii) Easy to program software

Various **disdvantages of CASE tools** are as follows :

(i) *Purchasing of case tools is not and easy task :* Its cost is very high. Due to this reason small software development firm do not invest in case tools.

(ii) *Learning curve :* In general cases programmer productivity may fall in initial phase of implementation as user need time to learn this technology.

(iii) *Tool mix :* It is important to make proper selection of case tools to get maximum benefit from the case tool, so wrong selection may lead to wrong result.

Q.9. Write notes on : (20)

(a) Software quality attributes,

(b) Software reliability.

Ans. (a) Software quality attributes : Refer Q.9(a) of paper May 2014.

(b) Software Reliabilty : "Software Reliability is defined as the probabilty of failure-free software operation for a specified period of time in a specified environment".

Informally, Software Reliability can be defined as a measure of how well the users think the system operators according to its specifications".

Jelinski-Moranda Model : The JM model is the best-known model of reliability models. This model is based in Markov process. The Jelinski-Moranda model says that the hazard rate is a step function, where improvements in reliability only takes place when a failure is fixed, and each failure contributes equally to the overall reliability. The plot of Rate Of Occurrence Of Failures (ROCOF) versus time for the Jelinski-Moranda Model is shown in Fig.

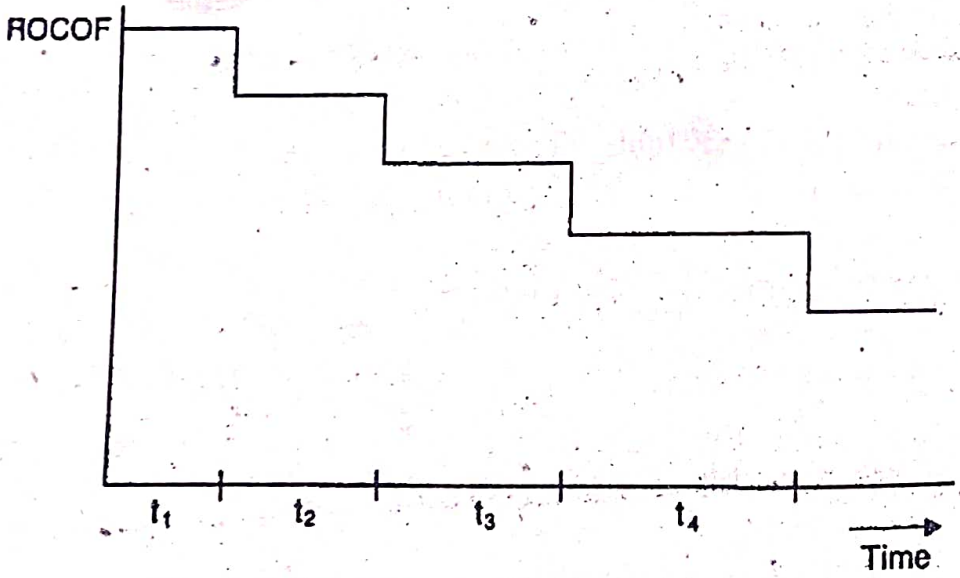


Fig : Jelinksi-Moranda Model ROCOF Plot

The ROCOF is defined as the probability of a failure, not necessarily the first, in a defined interval.

It assumes that for each i.

$$F_i(t_i) = 1 - e^{-\lambda_i t_i} \text{ with } \lambda_i = (N - i + 1)\phi$$

Here N is the initial number of faults, and f is the contribution of each fault to the overall

failure. The model also assumes that all faults have the same rate. The inference procedure of JM is called maximum likelihood estimation. For a particular set of failure data, this procedure produces etimates of N_i and ϕ_i. Then t_i is predicted by putting these estimates into the model.

Following are two important reliability models based on JM model :

(i) *Shooman's Reliability Model*

(ii) *Musa's Reliability Model*

These models are briefly discussed below.

Shooman's Reliability Model : Various reliability models based on JM have been produced. Shooman's model is also a similar model with some additional assumptions and postulates, such as :

– Hazard rate is proportional to the number of remaining errors.

– Hazards are detemined by the rate at which the remaining errors were passed in the execution of the program.

Thus, hazard rate depends on depends on the instruction processing rate, the size of the program and the number of errors remaining in the program.

Musa's Reliability Model : The Musa reliability model has JM model as its basis but builds features into it (Musa 1987). It is the first model to use execution time to gain inter-failure times. Musa decided that software reliability theory should be based on execution time rather than calendar time.

The execution time model is richer in simplicity and clarity of modelling and has better conceptual insight and predictive validity.

Musa model views execution time in two ways :

– Operating time of the software product and

– Cumulative execution time that occurs during the test phase fo development and the post delivery maintenance.

The hazard rate was assumed to be constant w.r.t. operating time, but it varies as a function of the errors remaining and hence cumulative execution time.

PRINCIPLES OF SOFTWARES

May 2016

Paper Code: CSE-302-F

Note: *Attempt five questions in all, selecting one question from each section. Q. No.* 1 *is compulsory.*

Q.1. Describe the following :

(a) Evolving role of software

(b) Information Hidding

(c) Debugging

(d) The ISO 9001 standard

Ans. (a) Evolving role of software : Software engineering principles have evolved over the past more than fifty years from art to an engineering discipline. It can be shown with the help of the following fig.

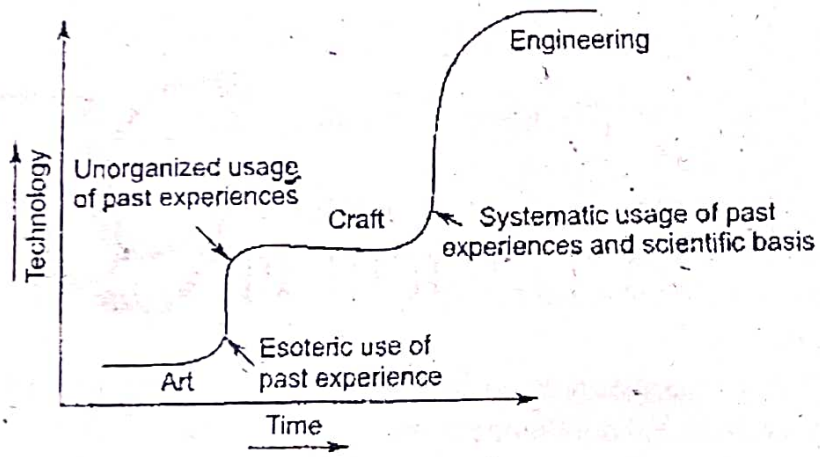


Fig. : Evolution of art to an engineering discipline

Development in the field of software and hardware computing make a significant change in the twentieth century. We can decide the software development process into four areas :

(i) Early Era : During the early eras general-purpose hardware became commonplace. Software, on the other hand, was custom-designed for each application and had a relatively limited distribution. Most software was developed and ultimately used by the same person or organization.

In the era the softawre are mainly based on (1950-1960)

– Limited distribution

– Custom software

– Batch Orientation

(i., Second Era : The second era to computer system evolution introduced new concepts of human machine interaction. Interactive techinques opened a new world of application and new levels of hardware and software sophistication. Real time software deals with the changing environment and one other is multi-user in which many users can perform or work on a software at a time.

In this era the software are mainly based on (1960-1972)

– Multi-user
– Data base
– Real time
– Product Software
– Multiprogramming

(iii) Third Era : In the earlier age the software was custom designed and limited distribution but in this era the software was consumer designed and the distribution is also not limited. The cost of the hardware is also very low in this era.

In this era the software are mainly based on (1973-1985)

– Embedded intelligence
– Consumer Impact
– Distributed Systems
– Low cost hardware

(iv) Fourth Era : The fourth era of computer system evolution moves us away from individual computers and computer programs and toward the collective impact of computers and software. As the fourth era progresses, new technologies have begun to emerge.

In this the software are mainly based on (1985-)

– Powerful Desktop systems
– Expert systems
– Artificial intelligence
– Parallel Computing
– Object oriented technology

At this time the concept of software making is object oriented technology or network computing etc.

Ans.(b)Information Hidding : In the application of the information-hiding principle, the data structure is not directly used by other modules; it is used only through access functions.

The advantages of information hiding and decoupling data structure from the processing module is that such a system is easy to maintain, be there is change in structure or process. That is, if the data structure changed, the modification is limited to the structure and access function, and modules are not affected at all. The same is true when the structure in left intact but the process needs to be changed.

In object-oriented programming, the principle of information hiding is extensively used. In this approach, information that is not needed in that module is hidden from the module. The advantage is that the module controls the data hidden into it. Other modules are not allowed to access or modify the data.

It is defined as information captured in the data structure should be hidden from the rest of system. If it is used and some data structure is changed, then its effect is limited to the access for change information hiding is supported by object oriented language.

Ans.(c)Debugging : Debugging means identifying, locating and correcting the bugs usually by running the program. It is an extensively used term in programming. These bugs are usually logical errors.

During compilation phase the source files is accessed and if errors are found, then that file is edited and the corrections are posted in the file. After the errors have been detected and the corrections have been included in the source file, the file is recompiled. This detection of errors and removal of those errors is called debugging. The file is compiled again, so changes done last time gets included in the objects file also by itself. This process of compilation, dedugging and correction posting in the source file contiunes until all syntactical errors are removed completely. If program is very large and complex, more is the number of times the program has to be corrected and compiled.

Successful compilation of the program means that now the program is following all the rules of the language and ready to execute. All of the syntax errrors of the program are indicated by the compiler at this stage.

Debugging Tactics/Categories : The various catagories for debugging are

(i) Brute force debugging
(ii) Backtracking
(iii) Cause elimination
(iv) Program slicing
(v) Fault tree analysis.

Ans.(d)ISO 9001 standard : ISO 9001 is a Quanlity assurance standard that is specific to software engineering. It specifies 20 standards with which an organization must comply for an effective implementaion of the quality assurance system. It is an international quality management syatem. ISO 9001 is mainly related to the software industry. It lay down the standards for designing, developing, servicing the producing of standard quality of goods. It is also applicable to most software development organizations.

The salient features of ISO 9001 requirements are as follows :

(i) All documents concerned with the development of a software product should be properly managed, authorized and controlled.

(ii) Proper plans should be prepared and then progress against these plans should be monitored.

(iii) Important documents should be independently checked and reviewed for effectiveness and correctness.

(iv) The product should be tested against its specification.

Section - A

Q.2. Give a complete description about software life cycle models.

Ans. There are various software development life cycle (SDLC) models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development.

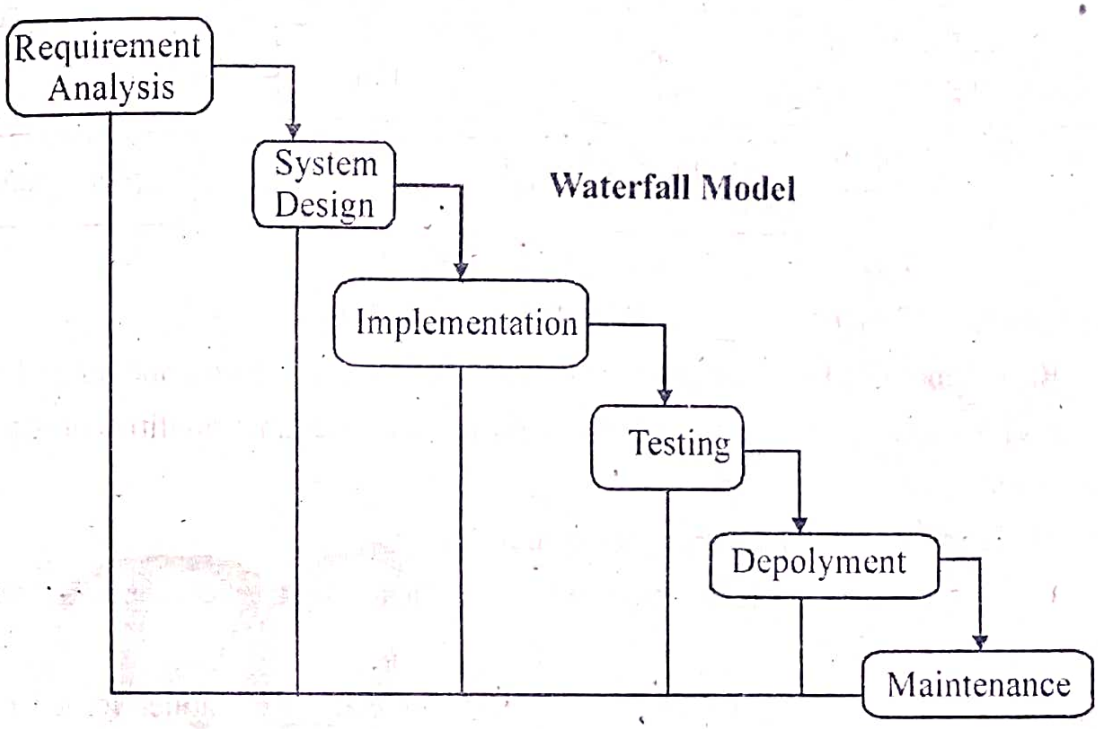
Following are the most important and popular SDLC models followed in the industry:

1. Waterfall Model
2. Iterative Model
3. Spiral Model

4. V-Model
5. Big Bang Model

The other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

1. SDLC Waterfall Model : Following is a diagrammatic representation of different phases of waterfall model.



Waterfall Model

The sequential phases in Waterfall model are:

Requirement Gathering and analysis : All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

System Design: The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

Implementation : With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

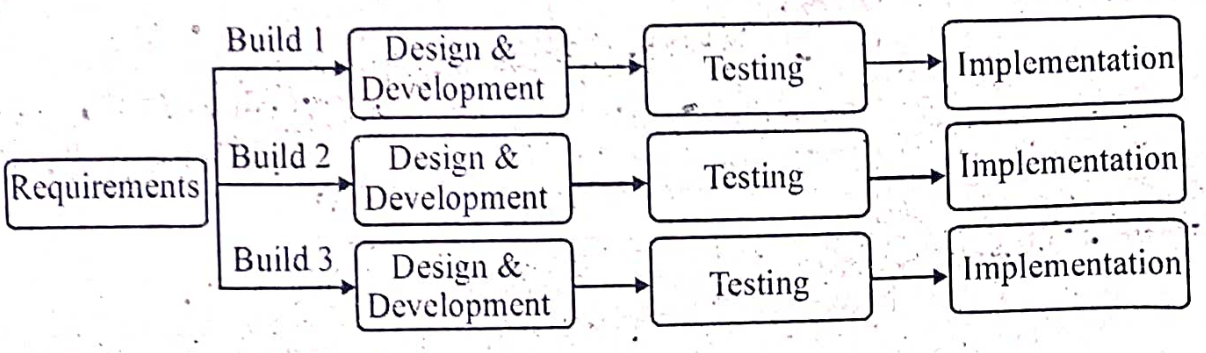
Integration and Testing : All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system : Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.

Maintenance : There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

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

2. SDLC Iterative Model : Following is the pictorial representation of Iterative and Incremental model:



This model is most often used in the following scenarios:-

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.

3. SDLC Spiral Model : The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification : This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

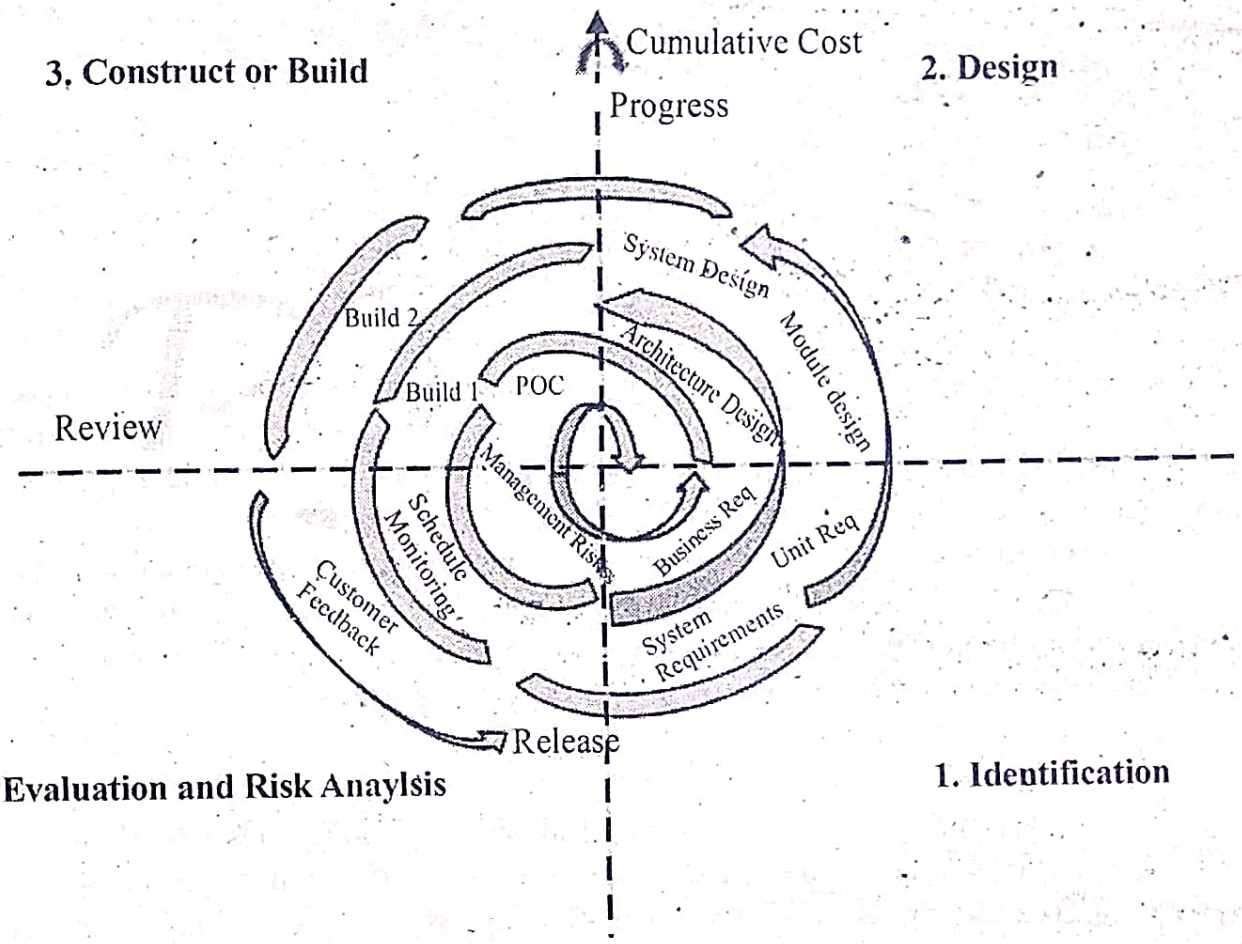
Design : Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct or Build : Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Evaluation and Risk Analysis : Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

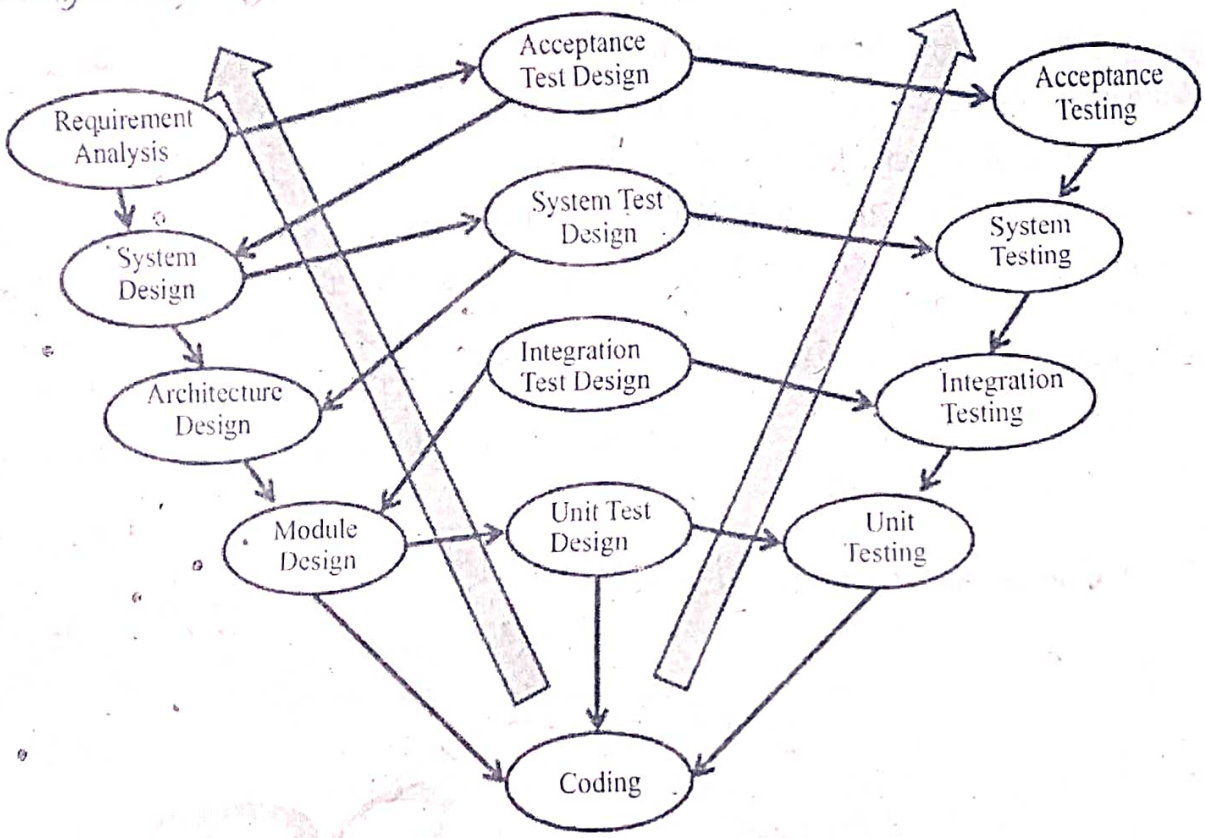
Following is a diagrammatic representation of spiral model listing the activities in each phase:



4. V Model : The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

The below figure illustrates the different phases in V-Model of SDLC.



5. SDLC Big Bang Model : The Big Bang model is SDLC model where there is no specific process followed. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement.

Big Bang Model is SDLC model where there is no formal development followed and very little planning is required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

Q.3.(a) What do you mean by Risk Analysis and Managment? Explain.

Ans. Risk Analysis : When the risks have been identified, all items are analyzed using different criteria. The purpose of the risk analysis is to assess to loss probability and magnitude of each risk item.

The input is the risk statement and context developed in the identification phase. The output of this phase is a risk list containing relative ranking of the risks and a further analysis of the description, probability, consequence and context (Hulett 1996). The main activities in this phase are :

(i) Group similar risks : Detect duplicates and find new risk items by grouping the identified risks into categories.

(ii) Determine risk drivers : The risk drivers are parameters that affect the identified risk. For example, schedule drivers are incided in the critical path model. Determining these properties help to assess and prioritize the risks.

(iii) Detemine source of risks : The sources of risks are the root causes of the risks. These are determined by asking the question why ? and trying of figure out what may have caused the risk. Several root causes may lead to the same risk.

(iv) Estimate risk exposure : The risk exposure is a measure of the probability and the consequence of a risk item. The consequence can also be stated in terms of loss (for example life, money, property, reputation).

(v) Evaluate aganist criteria : Each risk item is evaluated using the predefined criteria, which are important for the specific project. Criteria may be stated in terms of the probability occurrence, the consequence and the time frame. This information is used to prioritize the risks.

Once this is done, risks can be prioritised, and the most serious risks can be identified for monitoring.

Software Riks Management : Software risk management is the process of identifying software risks and planning to avoid those risks or to minimize the effects if they cannot be avoided.

Using risk management techniques, we can alleviate the harm or loss in a software project (Hall 1998). All risk should not be avoided, but by performing risk management, we can attempt to ensure that the right risks are taken at the right time. ".... risk taking is essential to progress and, and failure is often a key part of learning.

Risk management is a concept that can be implemented in a number of ways (Gemmer 1997). All good risk management approaches have the following characteritics:

- There is a planned and documented risk management process for the project of program;
- The process is based on a prospective assesment. The project management team looks ahead to find and manage possible problems;
- The initial assessment is peridically redone to validate the initial findings and to uncover new problem areas;
- The program has a defined set of evaluation criteria that covers all facets of the program;
- The on-going results of the risk management process are formally documented.

Q.3.(b) Give a complete description about Software Project Management.

Ans. Software Project Management : The goal of project management is to ensure that a software product that meets the quality specification of the requirements is produced and at an economicallly justifiable cost within the planned resources (e.g. personnel, equipment, tools, etc).

In other words, Project Management is a series of activities embodied in a process of getting things done on a project by work with members of the project team and with other people in order to reach the project schedule, cost and technical performance objectives.

Activities in Project Management : The primary tasks of project management are planning, organization and technical and economic control of project execution.

The following are the major activities in project managment:

- Proposal Writing
- Project planning
- Project Scheduling

- Project Tracking
- Personal selection and Evaluation
- Project Report Writing.

Proposal Writing is a skill that is acquired by experience. It is a critical task as the existence of many software organisations depends on having enough proposals accepted and contracts awarded. There can be no set guidelines for this task.

Project planning is concerned with identifying the activities, milestones and devliverables produced by a project. A plan must be drawn up to guide the development towards the project goals.

Project Scheduling is one of the most difficult tasks of project management. It involves sequarting the total work involved in a project into separate tasks and assessing when these tasks will the completed. Usually, some of these tasks are carried out in parallel. Project schedulers must coordinate these parallel tasks and organize the work so that the work force is used optimally.

Project tracking is a continuing project activity. The activity is also known as Project Monitoring. The managers must keep track of the progress of the project and compare actual and planned progress and costs. Most organisations employ formal mechanisms for monitoring whereas a skilled manager can often moniter the project activities by informal discussion with project staff.

Personal selection and evaluation is a very challenging task. Project manages usually have to select people to work on their project. Ideally, skilled staff with appropriate experience will be avaiable to work on the project. However, in most cases, managers have to settle for a less than ideal project team.

The project manager is usually responsible for project report writing to both the client and contractor organisations. Project managers must write concise, coherent, document that abstract critical information from detailed project reports.

Section - B

Q.4.(a) Explain in detail about classical Analysis methods.

Ans. Classical Analysis Methods : The classical analysis methods are systematic, but many safety process start with an informal stage, focusing on identifying (potential) hazards.

In many cases new systems are an evolution of previous systems, what is referred to as "normal design". In such cases, it is usual to identify harzards by reviewing the list from the previous system (s), and adjusting for any changes in design concept. This can be illustrated by considering aircraft landing gear: a typical hazard list would include.

- Failure to deploy prior to landing
- Depolyment in cruise.
- Asymmetric deloyment
- Failure to lock gear in deployed position.

Most landing gear will "castor", but are not activily steered. On the A380 the gear are activiely stered, so the typical hazards above would be supplemented by those that are specific to this new capability, e.g., "rotated (away from straight ahead) prior to landing" (intutively this will seek to "push" the aircraft laternally and/or lead to tire failure and the widespread distributions of high energy debris).

In general,harm to human being arises from toxic materails and or/uncontrolled energy. Design can be assessed for the presence of toxic materials-often the design will be modified to remove such materials,unless there are over-riding reasons for using them. In terms of energy, it is normal to consider energy types and what might happen if they are not controlled. Example energy types include.

- Potential -e.g. weigths at height.
- Kinetic - e.g. rotating machinery.,
- Pressure - e.g. hydraulies.
- Electrial - e.g. batteries.

Analysis of this sort will be by cosidering potential energy sources and looking "barries" which might contain the energy in the event fo failure, i.e., a deviation from design intent. Such controls might include restraints or harnesses for weights at height (to prevent or limit the drop), double-skinned hydraulic pipes, to contain leaks, in generate the barriers can either prevent a failure leading to a hazard, or an mitigate the hazard once it has occurred

Classfical analysis methods are used later in the development and safety process ; the investigate (hypothesized) faults or failures, exploring the properties of (proposed) system designs. The methods fall into three classes.

- Deductive - working "backwards" from a hypothesized event of state, e.g. hazard, to identtify its possible causes; probably the most widely used method is fault analysis (FTA);
- Inductive - working "forward" from a known or hypothesized event or state e.g. a component failure, to identify it's possible consequences, a widely used method is failure modes the effects analysis (FMEA) and its variants.
- Exploratory – working both forwards the backwards from a hypothesized even to identify cuases and consequences; Hazard and Operability Studied (HAZOP (S)) and its variants is probably the most widely used example of this class of analysis and is of particular relevence to software.

Q.4.(b) Give a brief description about Behavioural modeling

Ans. All behavioural models really do is describe the control structure of a system. This can be things like:

- Sequence of operations
- Object states
- and Object interactions

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

Representation of Behavioural modeling : So, how do we represent behavioural models ? Well, we represent them with dynamic diagrams. For example:

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

For consistency i will use communication diagram and state diagram.

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

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

Q.5. Explain in detail about software Design.

Ans. Software design : Software design is the activity of specifying the nature and composition of a software system satisfying client needs and desires, subject to design constraints".

Design principles : Principle in designing a software are as follows :

(i) Abstraction : Abstraction (separtes concepts from instantiations). It allows designers to focus on solving a problem without being concerned about irrelevant lower level details.

– Procedural abstraction - named sequence of events.

– Data abstraction - named collection of data objects.

(ii) Refinement : It's the process of elaboration where the designer provides successively more detail for each design component.

– Decompose design decisioin top – down to elementary level.

– Isolate design aspects that are not independent.

– Add details incrementally each step.

– Postpone decisions relating to detailed representations.

– Continually demonstrate that each refinement step is a correct expansion of previous steps.

(iii) Modularity : A module is a named entity that :

– Contains instructioins, processing logic, and data structures.

– Can be separately compiled and stored in a library.

– Can be included in a program.

– Module segments can be used by invoking a name and some parameters.

– Modules can use other modules.

Modularity is the degree to which software can be understood by examining its components independently of one another.

(iv) Software Architecture : Overall structure of the software components and the ways in which that structure provides conceptual integrity for a system.

Software architecture is a sketchy map of the system. Software architecture describes the coarse grain components (usually descibes the computation) of the system. The connectors between these components describe the communication, which are explicit and pictured in a relatively detailed way. In the implementation phase, the coarse components are refined into "actual components", e.g, classes, and Objects. In the object-oriented field, the connectors are usually implemented as interfaces, Control hieracrchy or program structure.

(v) Structural Planning : In *horizontal paritioning* the control modules are used to co-ordinate communication between and execution of the functions. Partitioning this way provides the following benefits :

(i) Results in software that is easier to test and maintain.

(ii) Results in fewer propagation side-effects.

(iii) Results in software that is easier to extend.

In *vertical partitioning* the control (decision making) modules are located at the top, and work is distributed top-down in the program structure. That is, top level functions perform

control functions and do little actual processing, while modules that are low in the structure perform all input, computation and output tasks. As changes to programs usually revolves around one of these three tasks there is less likelihood that changes made to the lower modules will propagate (upwards) making this partitioning strategy more maintainable.

(vi) Data Structure : It's the representation of the logical relationship among individual data elements. (requires at least as such attention as algorithm design).

Software Procedure : The precise specification of processing :

– Event sequences.

– Decision points.

– Repetitive operations.

– Data organization/structure.

(vii) Information Hiding : The principle of information hiding is the hiding of desing decision in a computer program that are most likely to change, thus protecting other parts of the program from change if the desing decision is changed. Information (data and procedure) contained within a module is inaccessable to modules that have no need for such information. It should be usedto guide architectural design interface designs, and modularization. Modules hide difficult or changeable desing decisions.

Section -C

Q.6.(a) What do you understand by Architectural Design ? Explain about the architectural complexity.

Ans. Architectural Design : Architectural design is the specification of the major components of a system, their responsibilities, properties, interface, and the relationship and interactions between them. In architectural design the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design include:

- Gross decomposition of the system into major components;
- Allocation of functional responsibilities to components;
- Component interfaces;
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth,
- Communication and interaction between components.

Aspects of a system's architecture may be specified in its requirement but this is less often the case than with interface design.

Architectural design adds important details ignored during interface design. Design of the internals of the major components is ignorted until the last phase of design.

Architectural Complexity : A useful technique for assessing the overall complexity of a proposed architecture is to consider dependencies between components within the architecture. These depedencies are driven by information/control flow within the system. Zhao (ZHA98) suggests three types of dependencies.

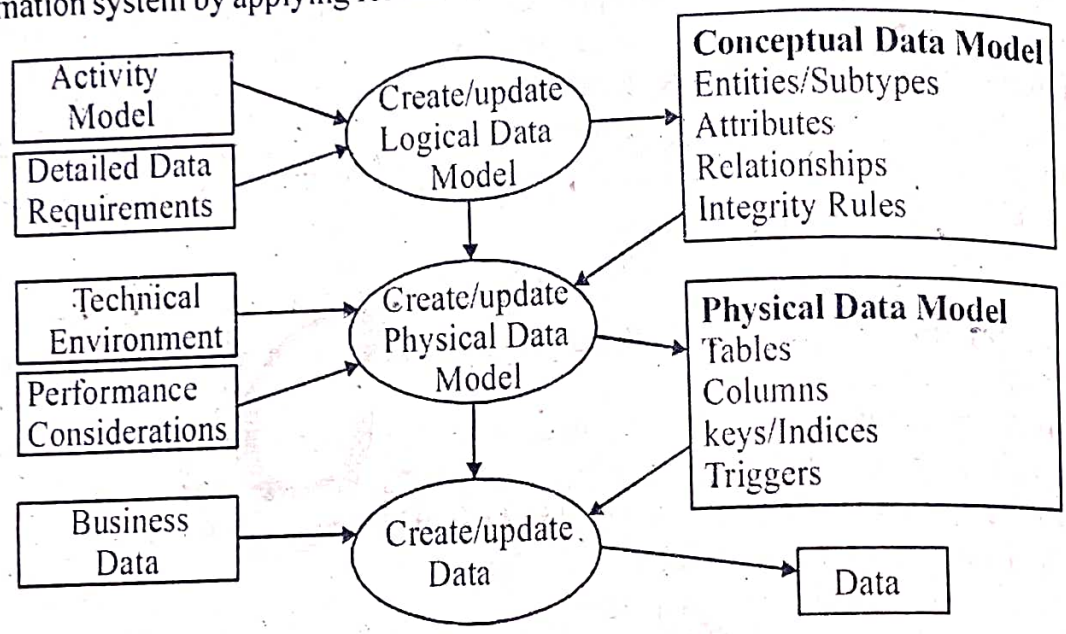
Sharing dependencies represent dependence relationship among consumers who use the same resource or producers who produce for the same consumer. For example, for two components u and v, if u and v refer to the same global data, then there exists a shared dependence relationship between u and v.

Flow dependencies represent dependence relationship between producers and consumers of resources. For example, for two components u and v, if u must complete before control flows into v (prerequisite), of if u communicates with v by parameters, then there exists a flow dependence relationship between u and v.

Constrained dependencies represent costraints on the relative flow of control among a set of activities. For example for two components u and v, if u and v can not execute at the same time (mutual exclusion), then there exists a constrained dependence relationship between u and v.

Q.6.(b) Define the term Data modeling.

Ans. Data modeling in software engineering is the process of creating a data model for an information system by applying formal data modeling techniques.



The data modeling process. The figure illustrates the way data models are developed and used today. A conceptual data model is developed based on the data requirements for the application that is being developed, perhaps in the context of an activity model. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects. This is then used as the start point for interface or database design.

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Therefore, the process of data modeling involves professional data modelers working closely with business stakeholders, as well as potential users of the information system. There are three different types of data models produced while progressing from requirements to the actual database to be used for the information system. The data requirements are initially recorded as a conceptual data model which is essentially a set of technology independent specifications about the data and is used to discuss initial requirements with the business stakeholders. The conceptual model is then translated into a logical data model, which documents structures of the data that can be implemented in databases. Implementation of one conceptual data model may require multiple logical data models. The last step in data modeling is transforming the logical data model to a physical data model that organizes the data into tables, and accounts for access, performance and storage details. Data modeling defines not just data elements, but

also their structures and the relationships between them.

Data modeling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modeling standards is strongly recommended for all projects requiring a standard means of defining and analyzing data within an organization, e.g., using data modeling:

(a) to assist business analysts, programmers, testers, manual writers, IT package selectors, engineers, managers, related organizations and clients to understand and use an agreed semi-formal model the concepts of the organization and how they relate to one another

(b) to manage data as a resource

(c) for the integration of information systems

(d) for designing databases/data warehouses (aka data repositories)

Data modeling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business. The data models should ideally be stored in a repository so that they can be retrieved, expanded, and edited over time. Whitten et al. (2004) determined two types of data modeling:

(i) Strategic data modeling: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for information systems is defined. Information engineering is a methodology that embraces this approach.

(ii) Data modeling during systems analysis: In systems analysis logical data models are created as part of the development of new databases.

Data modeling is also used as a technique for detailing business requirements for specific databases. It is sometimes called database modeling because a data model is eventually implemented in a database.

Q.7. Explain in detail about software Testing.

Ans. *Software testing* is the process of executing a program with the intention of finding errors in the code. It is the process of exercising or evaluating a system or system component by manual or automatic means to verify that it satisfies specified requirements or to identify diffenences between expected and actual results.

The objective of testing is to show incorrectness and testing is considered to succeed when an error is detected. An error is a conceptual mistake made by either the programmer or the designer or a discrepancy between a computed value and a theoretically correct value.

The objectives of software testing are listed below :

(i) The main objective of software testing to execute a program in a proper way, where there should be error arises during execution time.

(ii) A good test is one that has a high probability of finding as yet undiscovered errors.

(iii) To deliver high quality of software project.

(iv) To ensure that sofrware is accroding to the requirement of the users.

(v) To ensure that sofrware is reliable and working as per the specificaion.

(vi) System actually uncover different type errors in minimum time and with a minimum amount effors.

(vii) It also help in minimize cost of maintenance, if software project is tested in systematic way.

(viii) It also help in minimize chances of failure after its implementation.

(ix) It also help in reducing catastrophic failure, which is very dangerous, if software fail during execution time. E.g. Aircraft Control, Nuclear Reactor Control Plant.

(x) Software Quality Improvement.

(xi) Verification and validation.

(xii) Software Reliability Estimation.

(xiii) A Successful test is one, in which there is no errors are founds, our objective is to design test that systematically uncover different classes of errors and do with a minimum amount of time and effort.

Testing Principles :

(1) Testing should be based on user requirements : This is in order to uncover any defects that might cause the program or system to fail to meet the client's requirements.

(2) **It is impossible to test everything :** Exhaustive tests of all possible scenarios are impossible, simple because of the many different variables affecting the system and the number of paths a program flow might take.

(3) Use effective resources to test : This represents use of the most suitable tools, procedures and individuals to conduct the tests. The test team should use tools that they are confident and familiar with. Testing procedures should be clearly defined. Testing personnel may be a technical group of people independent of the development.

(4)Testing planning should be done early : This is because test planning can begin independently of coding and as soon as the client requirements are set.

(5) Test for invalid and unexpected input conditions as well as valid condtions : The program should generate correct messages when an invalid test is encountered and should generate correct results when the test is valid.

(6) Testing should begin at the module : The focus of testing should be concentrated on the smallest programming units first and then expand to other parts of the system.

(7) Testing must be doen by an independent party : Testing should not be performed by the person or team that developed the software since they tend to defend the correctness of the program.

(8) Assign best personnel to the task : Because testing requires high creativity and responsibilty only the best personnel, must be assigned to design, implement, and analyze test cases, test data and test results.

(9) Testing is the process of executing software with the intent of finding errors.

(10) Keep software static during test : The program must not be modified during the implementation of the set of designed test cases.

(11) Document test cases and test results.

(12) Provide expected test results if possible : A necessary part of test documentation is the specification fo expected results, even if providing such results is impractical.

Section - D

Q.8. Explain the foliowing:

(i) Software Reviews.

(ii) Software Reliability

Ans. (i) Software Reviews : A review can be defined as : "A meeting at which the software elements is presented to project personal, managers, users customers or other interested partics for comment or a approval".

OR

"A software review process can be defined as a critical evaluation of an object".

OR

A review is a process or meeting during which a work product, or a set of work products, is presented to project personnel, managers, users customers, or other intersted parties for comment or approval.

OR

"A process or meeting during a software product is lexamined byl project managers, users, customers, users representatives, or other interrested parties for comment or approval".

In this context, the term "software product" means "any technaical document or partical document producted as a deliverable of a software development activity", and may include documents such as contracts, project plans and budgets, requirements documents, specifications, designs, source code, users documentation, supports and maintenance documentation, test plans test specifications, standdards, and any other type of specialist work product.

Purpose of software Review : The purpose of any review is :

- To discover errors in analysis, design and coding testing and implementation phase of software development cycle.
- To see whether procedures are applied uniformly and in a manageable manner.

The reviews predominantly are at three stages ; first, Requirement Review, then Design Review and finally Technical Review. Every organization has review guidelines and there is a set procedure for each review. The review is performed by an authorized person, review meetings are held, as issues list is made, and descisions and actions are taken to resovle them. The issues and the solutions are rechecked in the next meeting. Table shows the main reviews and the focus in each of them.

Table : Main reviews, its participants and their focus

Review	*Participants*	*Focus*
Requirement	Customer, Project Manger, Users	Meeting the Requirment and software specification
Design	Project manager, system designer, solution architect, and customer	Architecture, Performance, Deliverables.
Technical	Project manager, designer, solution Architect.	Quality, Standards and Deliverables

Objectives for Software Reviews : Reviews objectives are :

- To ensure that the software element conforms to its specifications.
- To ensure that the development of the software element is being done as per plan standards are guidelines applicable for the project.
- To ensure that the changes to the software element property implemented and affect only those system areas identified by the change specifications.

Varieties of Software Reviews : Software reviews may be divided into three categories.

- Software per reviews are conducted by the author of the work product, or by one or more colleagues of the author, to evaluate the technical content and/or quality of the work.
- Software management reviews are conducted by management representatives to evaluate the status of work done and to make decision regarding downstream activities.
- Software audit reviews are conducted by personnel external to the software project, to evaluate compliance with specifications, standards contractual agreements, or their criteria.

Ans.(ii)Software Reliabilty : "Software Reliability is defined as the probabilty of failure-free software operation for a specified period of time in a specified environment".

Informally, Software Reliability can be defined as a measure of how well the users think the system operators according to its specifications".

Jelinski-Moranda Model : The JM model is the best-known model of reliability models. This model is based in Markov process. The Jelinski-Moranda model says that the hazard rate is a step function, where improvements in reliability only takes place when a failure is fixed, and each failure contributes equally to the overall reliability. The plot of Rate Of Occurrence Of Falilures (ROCOF) versus time for the Jelinski-Moranda Model is shown in Fig.

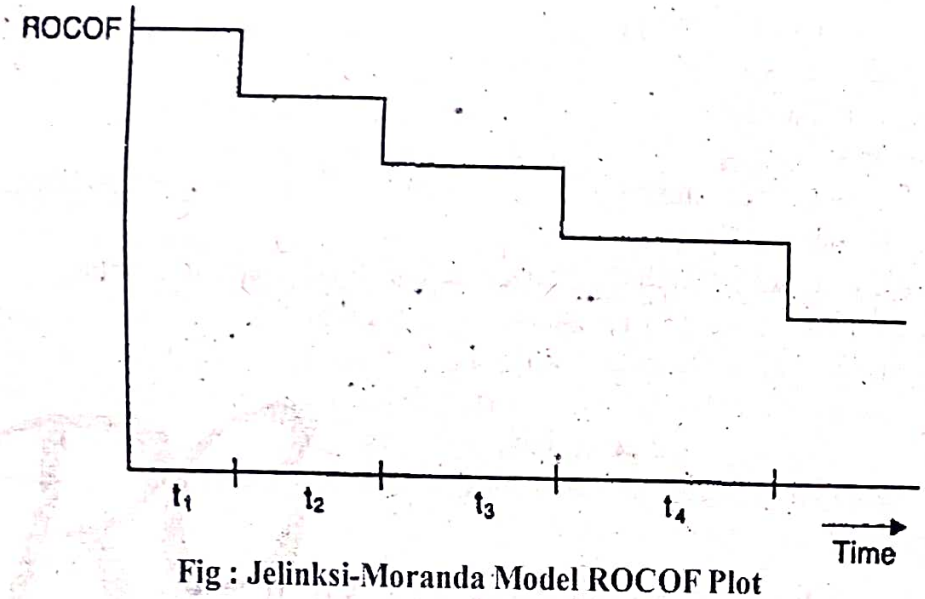


Fig : Jelinksi-Moranda Model ROCOF Plot

The ROCOF is defined as the probability of a failure, not necessarily the first, in a defined interval.

It assumes that for each *i*.

$$F_i(t_i) = 1 - e^{-\lambda_i t_i} \text{ with } \lambda_i = (N - i + I)\phi$$

Here N is the initial number of faults, and f is the contribution of each fault to the overall failure. The model also assumes that all faults have the same rate. The inference procedure of JM is called maximum likelihood estimation. For a particular set of failure data, this procedure produces etimates of N_i and ϕ_i. Then t_i is predicted by putting these estimates into the model.

Following are two important reliability models based on JM model :

(i) *Shooman's Reliability Model*

(ii) *Musa's Reliability Model*

These models are briefly discussed below.

Shooman's Reliability Model : Various reliability models based on JM have been produced. Shooman's model is also a similar model with some additional assumptions and postulates, such as :

– Hazard rate is proportional to the number of remaining errors.

– Hazards are detemined by the rate at which the remaining errors were passed in the execution of the program.

Thus, hazard rate depends on depends on the instruction processing rate, the size of the program and the number of errors remaining in the program.

Musa's Reliability Model : The Musa reliability model has JM model as its basis but builds features into it (Musa 1987). It is the first model to use execution time to gain inter-failure

times. Musa decided that software reliability theory should be based on execution time rather than calendar time.

The execution time model is richer in simplicity and clarity of modelling and has better conceptual insight and predictive validity.

Musa model views execution time in two ways :

– Operating time of the software product and

– Cumulative execution time that occurs during the test phase fo development and the post delivery maintenance.

The hazard rate was assumed to be constant w.r.t. operating time, but it varies as a function of the errors remaining and hence cumulative execution time.

Q.9. Explain the following :

(i) Statistical software Quality Assurance.

(ii) Computer Aided software engineering.

Ans. (i) Statistical software Quality Assurance : Statistical quality assurance techniques have helped some softwarer organizations achieve a 50% defect reduction annually. Statistical quality assurance reflects growing trend throughout the industry to become more quantitative about quality. For software engineering, statistical quality assurance implies the following steps:

(i) Information about software errors and defects is collected and categorized.

(ii) An attempt is made to trace earh error and defect to its underlying cause.

(iii) Using the Parato, principle (80% of the defects can be traced to 20% of all possible causes), isolate the 20 % (vital few).

(iv) Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects.

This relatively simple concept represents an important step toward the creation of an adaptive software process in which changes are made to improve those elements of the process that introduce error.

Six Sigma is one approach to product and process improvement that has gained acceptance in many industries globally, including those focused on software development. The six sigma methodology makes use of statistical techniques to improve process capability and reduce product defects. Defects are defined as any product attributes that are outside customer expectations. Removing these defects has the potential to increase the level of customer satisfaction. The metrics chosen are selected because they are aligned with the customer's goals for the products. The products is scored using these metries. When metric values are not acceptably close to their predetermined values, the software developers suspect a defect may be present. Root cause analysis is undertaken to determine the process weakness that caused the product defect. The goal is to produce product zero defects.

There is a software engineering specialization of the Six Sigma approach known as Software Six Sigma. Software Six Sigma is based on three principles focus on customers, process orientation, and leadership, based on metrics. It is a management strategy which relies on defect metrics as its main tool to reduce costs and improve customer satisfaction. Before using Software Six Sigma developerss must know their customers needs. The cost savings largely result from avoiding rework caused by delivering defective products and reduced customer satisfication. The metrics used to mointor defects need to include quality factors that affect the UX (e.g., usability or reliability) as well as counting thing like code smells. QFD is often used to keep the development team focused on customer quality goals.

The steps for using Six Sigma for process improvment (DMAIC) are lised below.

- Set the goal – *Define*
- Define the metrics – *Measure*
- Measure where you go – *Analyze*
- Improve processes while you go – *Improve*
- Act immediately if going on the wrong path – *Control.*

Six Sigma helps to speed up the test and integration parts of product development. It can be used to continuously improve processes and product quality, based on attributes of software products under development. It is an effective project management tool that can provide software engineers with tools to make data-based decisions. In some cases this process may allow engineers to predict the occurrence of defects before they occur based on their prevalence.

There some reasons why statistical techniques for SQA have not been accepted by software developers. Some developers believe Six Sigma techniques are too complicated or too costly to use on routine projects. Some developers believe that software development does not follows standard process like those present, I manufacturing. Some developers do not grasp the connection between process improvement and product improvement by reducing the points where defects are injected into the product.

Ans.(ii) Computer Aided Software Engineering : CASE stands for Computer Aided Software Engineering.

CASE is a total which aids a software engineer to maintain and develop software. The workshop for software engineering is called an intergrated project support Enviroment (IPSE) and the tools set that fills the workshop is called CASE.

CASE is a computer aided software engineering technology CASE is an automated support tool for the software engineers in any software engineering process.

Software engineers in any software engineering process.

- Translation of user needs into software requirements
- Transaction of software requirements into design specification
- Implementation of desgin into code
- Testing of the code
- Documentation

CASE technology provides software process supports by automating some process activities and by providing information about the software, which is being developed. Examples of activities, which can be automated using CASE, include:

- The development of graphical system models as parts of the requirements specification or the software design.
- Understanding a design using a data dictionary, which holds information about the entities and relations in a design.
- The generation of user interfaces from a graphical interface description, which is created interactively by the users.
- Program debugging through the provision of information about an executing program..
- The automated translation of programs from an old version of a programming language such a COBOL to a more recent version.

The use of computer Aided software Engineering (CASE) tool reduce the effort of developement of achieving quality goals and managing change and configuration throughout the product life cycle. It also help the project manager, the software developer and other key personal to improve their productivity in the development team.

PRINCIPLES OF SOFTWARE ENGG.

May - 2017

Paper Code:-CSE-302-F

Note : *Attempt five questions in all, selecting one question from each Section. Question No.* 1 *is compulsory. All questions carry equal marks.*

Q.1. Explain the following : (20)

(a) Software products

(b) Design Documentation.

(c) Restructuring.

(d) SQA

Ans. (a) Software products : Software Product are nothing but software systems delivered to customer with the documentation that describes how to intall and use the system. In certain cases, software products may be part of system products where hardware as well as software is delivered to a customer.

Types of software products : Software products fall into two broad categories :

Generic Products : These are stand-alone system that are produced by a development organisation and sold on the open market to any customer who is able to buy them.

Customised products : These are system that are commissioned by a particular customer. Some contractor develops of software for that customer.

Ans.(b)Design Documentation : IEEE defines software design documentation as 'a description of software created to facilitate analysis, planning, implementation, and decision-making. This design description is -used as a medium for communicating software design information and can be considered as a blueprint or model of the system.

There are two types of design documents, namely, high-level design document and low-level design documents.

A high level design document contains architectural details.

A low-level design document contains the data structure and its associated components along with interaction details.

Ans.(c)Restructuring : Software restructuring is a form of perfective maintenance that modifies the structure of a program's source code. Its goal is increased maintainability to better facilitate other maintenance activities, such as adding new functionality to, or correcting previously undetected errors within a software system. Changes to the structure are introduced through the application of transformations. Manually transforming the source code may introduce undesirable as well as undetectable changes in the system's behaviour. It is very difficult to ensure that manual restructuring preserves functionality and guaranteeing it is almost impossible. The problems associated with manual restructuring can be addressed by using a restructuring tool to automatically apply transformations. The majority of restructuring tools apply transformations by manipulating abstract program representations and specify the conditions of the transformation in terms of the representation structure. The context entity graph as program

representation was developed to support specific language constructs, but can be adapted to support a variety of programming languages. The implementation of a code abstraction transformation in terms of this structure is examined and various improvements are also suggested.

Ans.(d)SQA : The aim of the software Quality Assurance (SQA) process is to develop high-quality software product. Software Quality Assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained.

Quality assurance is a planned and systematic pattern of all actions necessary to provide adequate confindence that the item or product conforms to established requirements.

The purpose of a software quality assurance group is to provide assurance that the procedures tools, and techniques used during product development and modification are adequate to provide the desired level of confidence in the work products.

The process of software Quality Assurance : (i) Defines the requirements for software controlled system fault/failure detection, isolation, and recovery.

(ii) Reviews the software developement process and products for software error prevention and/or controlled change to reduced functionality states.

(iii) Defines the process for measuring and analyzing defects as well as reliability and main tainability factors.

SQA Objectives : The various objectives of SQA are as follows :

(i) Quality management approach.

(ii) Measurement and reporting mechanisms.

(iii) Effective software engineering technology.

(iv) A procedure to assure compliane with software development standards where applicable.

(v) A multi-testng strategy is drawn.

(vi) Formal technical reviews that are applied throughout the software process.

SQA Goals : The major goals of SQA are as follows :

(i) SQA activities are planned.

(ii) Non-compliance issues that cannot be resolved within the software project are addressed by senior management.

(iii) Adherence of software products and activities to the applicable standards, procedures and requirements is verified objectively.

(iv) Affected groups and individuals are informed of SQA activities and results.

Section - A

(20)

Q.2. Describe the following :

(a) Software crisis

(b) Software characteristics

Ans. (a) Software crisis : The *software crisis* is characterized by *an inability to develop software on time, on budget and within requirements*. As a result, the delivered software systems are :

(i) Completely unsatisfactory (not as per the specification)

(ii) Extremely late.

(iii) Far over the budget.
(iv) Poorly suited for intended user of the system.
(v) Projects running over-time.
(vi) Software was of low qualtity.
(vii) Software often did not meet requirements.
(viii) Projects were unmanageable & code difficult to maintain.
Various **reason of crisis** are as follows :
(i) Lack of communication between software developers and users.
(ii) Increase in cost of software compared to hardware.
(iii) Increase in the size of software.
(iv) Increased complexity of the porblem area.
(v) Project management problem.
(vi) Lack of understanding of the problem and its enviornment and
(vii) High optimistic estimates regarding software development time and cost
(viii) Duplication of efforts due to absent of automation in most of the software development activites.

Ans.(b) Software characteristics : The key characteristics of software are as under:
(i) Software does not wear out: There is a well-known "bath tub curve" in reliability studies for hardware products. The curve is given in Fig.(1). The shape of the curve is like "bath tub"; and is known as bath tub curve.

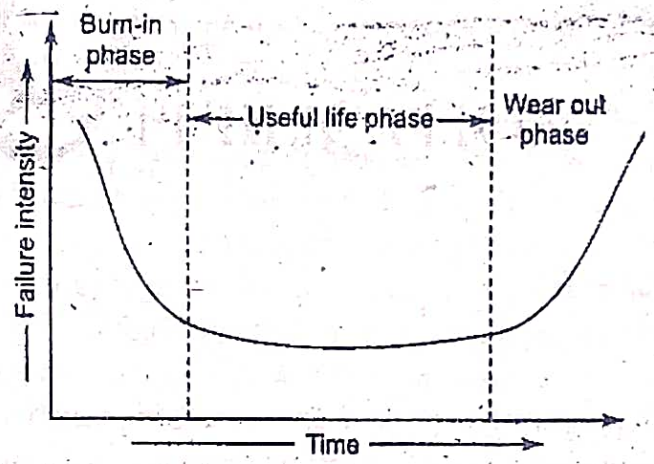


Fig.(1) : Bath Tub Curve.

There are three phases for the life of a hardware product. Initial phase is burn-in phase, where failure intensity is high. It is expected to test the product in the industry before delivery. Due to testing and fixing faults, failure intensity will come down initially and may stabilise after certain time. The second phase is the useful life phase where failure intensity is approximately constant and is called useful life of a product. After few years, again failure intensity will increase due to wearing out of components. This phase is called wear out phase. We do not have this phase for the software as it does not wear out. The curve for software is given in Fig.(2).

Important point is software becomes reliable overtime instead of wearing out. It becomes obsolete, if the environment for which it was developed, changes. Hence software may be retired due to environmental changes, new requirements, new expectations, etc.

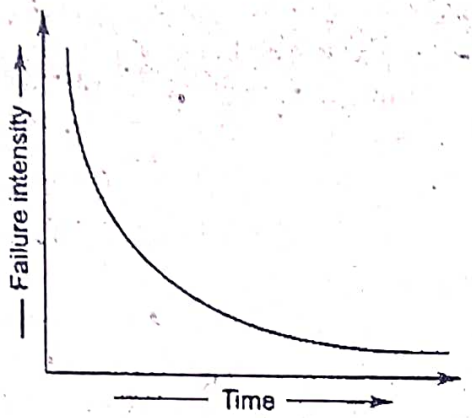


Fig.(2) : Software Curve.

(ii) 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 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.

(iii) Reusability of components: If we have to manufacture a TV, we may purchase picture tube from one vendor, cabinet from another, design card from third and other electronic components from fourth vendor. We will assemble every part and test the product thoroughly to produce a good quality TV. We may be required to manufacture only a few components or no component at all. We purchase every unit and component from the market and produce the finished product. We may have standard quality guidelines and effective processes to produce a good quality product. In software, every project is a new project. We start from the scratch and design every unit of the software product. Huge effort is required to develop a software which further increases the cost of the software product. However, effort has been made to design standard components that may be used in new projects. Software reusability has introduced another area and is known as component based software engineering.

Hence developers can concentrate on truly innovative elements of design, that is, the parts of the design that represent something new. As explained earlier, in the hardware world, component reuse is a natural part of the engineering process. In software, there is only a humble beginning like graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.

(iv) 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 accomodate any kind of change. However, most of the times, this "almost anything" characteristic has made software development difficult to plan, monitor and control. This unpredictability is the basis of what has been referred to for the past 30 years as the "Software Crisis".

Q.3.(a) Give a complete description about COCOMO. **(10)**

Ans. COCOMO model : COCOMO stands for *Constructive Cost Model*. It was introduced by Barry Boehm in 1981. It is perhaps the best known and most throughly documented of all software cost estimation models. It provides the following three levels of models :

(i) *Basic COCOMO model :* The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions :

$$\text{Effort} = a_1 \times (KLOC)^{a_2} \quad PM$$

$$T_{dev} = b_1 \times (Effort)^{b_2} \quad Months$$

Where, *KLOC* is the estimated size of the software product expressed in kilo lines, Code a_1, a_2, b_1, b_2 are constants of software products.

T_{dev} is the estimated time to develop the software, expressed in months.

Effort is the total effort required to develop the software product, expressed in person-month (PM).

Person months curve is shown in fig.(1).

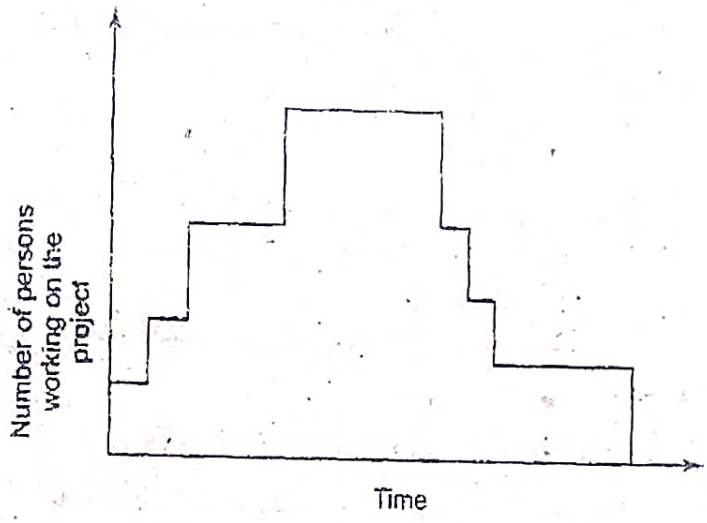


Fig.(1) : Person-month curve

Estimation of Development effort :

Organic: Effort = $2.4(\text{KLOC})^{1.05}$ PM

Semidetached : Effort = $3.0(\text{KLOC})^{1.12}$ PM

Embedded : Effort = $3.6(\text{KLOC})^{1.20}$ PM

Fig.2(a) shows a plot of estimated effort versus size for various product sizes.

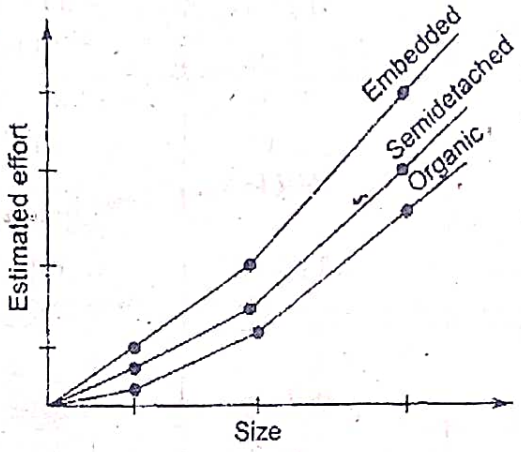


Fig.2(a) : Effort versus size

From fig.2(a), we can observe that the effort is almost linearly proportional to the size of the software product.

Estimation of Development time :

Organic: $T_{dev} = 2.5(\text{Effort})^{0.38}$ Months

Semidetached: $T_{dev} = 2.5(\text{Effort})^{0.35}$ Months

Embedded: $T_{dev} = 2.5(\text{Effort})^{0.32}$ Months

Fig.2(b) shows the plot is development time versus product size.

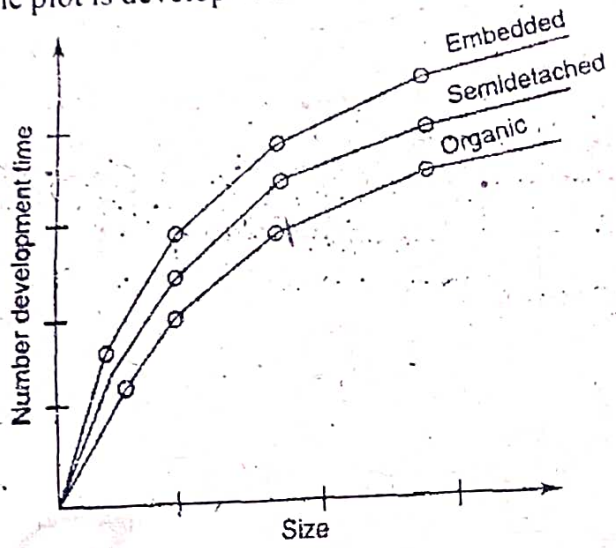


Fig.2(b) : Development Time versus size

From fig.2(b), we can observe that the development time is sub linear function of the size of the product.

(ii) *Intermediate COCOMO Model :* The intermediate COCOMO model recognizes this fact and refines the intial estimate obtained through the BASIC COCOMO expression by using a set of 15 cost drivers (multipliers) based on various attributes of software development.

Table : COCOMO Intermediate Cost drivers

Driver type	Code	Cost Driver
Product attributes	RELY	Required software reliability
	DATA	Database size
	CPLX	Product complexity
Computer attributes	TIME	Execution time constraints
	STOR	Main storage constraints
	VIRT	Virtual machine volatility-degree to which the operating system changes
	TURN	Computer turn around time
Personnel attributes	ACAP	Analyst capability
	AEXP	Application experience
	PCAP	Programmer capability
	VEXP	Virtual machine (i.e.,operation system) experience
	LEXP	Programming language experience

Project attributes	MODP	Use of modern programming practices
	TOOL	Use of software tools
	SCED	Required development schedule.

(iii) *Complete COCOMO Model :* The short-comings of both basic and intermediate COCOMO models are that they :

– Consider a software product as a single homogeneous entity.

– However, most large systems are made up of several smaller sub-systems. Some sub-systems may be considered as organic type, some embedded, etc. For some the reliability requirements may be high and so on.

– Cost of each sub-system is estimated separately.

– Costs of the sub-systems are added to obtain total-cost.

– Reduces the margin or error in the final estimate.

A Large amount of work has been done by Boehm to capture all significant aspects of a software development. It offers a means for processing all the project characteristics to construct a software estimate.

Q.3.(b) What do you mean by project scheduling and Tracking. Explain. (10)

Ans. Software Project Scheduling : Software Project Scheduling is one of the most difficult jobs of the software project manager. It is an extremely difficult because much of the information used in it is empirical and based on a particular individual's experience.

There are many techniques for estimating the time and resources required for a project (Mall 1999, Sommerville 1998, Pressman 1997). Estimates of the time and resources required for a development project are almost always optimistic, especially when the system being developed is technologically advanced or is significantly different from other projects that the organization has developed.

Additionally, competition for bids can lead to optimistic estimates. This is not unique to software development projects, but is more frequently true for software projects than other kinds of engineering projects.

Project Scheduling Activities : Project scheduling involves the following activities :

- Identify all the tasks and activities within the project
- Identify dependencies among the activities i.e. a unit must be tested before it can be integrated into the system.
- Estimate the resources (time) required for each activity
- Allocate people to activities
- Estimate the resources (time) required for the entire development project.

Section- B

Q.4.(a) Define prototyping. Explain in brief about prototyping method and tools. (10)

Ans. Prototyping offers an alternative approach that results in a demonstrable model of the software from which requirements can be refined. The prototype paradigm can be either closed-ended or open ended.

The closed ended approach is often called *throwaway prototyping*. Using this approach, a prototype serves solely as a rough demonstration of requirements. It is then discarded and the software is engineered using a different paradigm.

An open-ended approach called *evolutionary prototyping* uses the prototype as the first part of an analysis activity that will be continued into desgin and construction. The prototype of the software is the first evolution of the finished system. Before prototyping, it is necessary to determine whether the system to be built is amendable to prototyping.

Prototyping Technoques and Tools : For software prototyping to be effective, a prototype must be developed repidly so that the customer may assess results and recommend changes. To conduct rapid prototying, three generic classes of method and tools are available; fourth generation languages, reuse software components, formal specification/prototyping environments and languages (Very high level)

(a) *Fourth Generation Languages* : 4GL encompass a broad array of data base query and reporting languages, program and application generators and other high level non-procedural languages. As 4GL enable the software engineer to generate executable code quickly, they are ideal for rapid prototying. It is widely used for developing applications in the business domain as it involves updating a database and producing reports from the information held in the database. Standard forms are used for input and output. Database query language sueh as SQL is an example of 4GL. 4GLs may also package a report generator and a screen form designer with the quary language to provide a powerful environment for application development. 4GLs are generally used in conjunction with CASE tools for the development of small to medium sized systems. However, 4GLs are slower than conventional programming languages and require more resources like memory, time etc.

(b) *Reusable Components* : Another approach to rapid prototying is to assemble rather than build the prototype by using a set of existing software components. A software component may be a data structure or program module. However, a library system is to be developed so those components that exist can be catalogued and retrieved.

Developing prototypes with reusable components involves developing system specification by taking account of available reusable components. The components are taken from the 'component respository', put together to from the prototype system. This approach is usually suitable for throwaway prototyping.

(c) *Formal Specification and Prototyping Environments* : A number of formal specification languages and tools have been developed for natural language specification. These tools help an analyst to interactively create a language-based specification of a system or software, invoke automated tools that translate language based specifications into executable code and finally, the customer to refine the formal requirements after the demonstration.

(10)

Q.4.(b) Describe the following :

(i) Data flow diagram

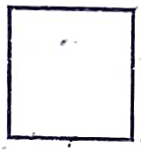
(ii) Data Dictionary

Ans.(i) Data flow diagram : Data Flow Diagram is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output.

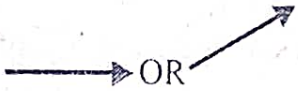
There are different types of symbols used to construct DFDs. The meaning of each symbol is explained below:

(1) Function symbol : A function is represented using a circle. This symbol is called a process or a bubble or performs some processing of input data.

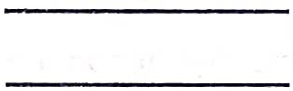
(2) External entity : A square defines a source of destination of system data. External entities represent any entity that supplies or receives information from the system but is not a part of the system.



(3) Data flow symbol : A directed arc or arrow is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.



(4) Data store symbol : A data store symbol is represented using two parallel lines. A logical file can represent either a data store symbol, which can represent either a data structure, or a physical file on disk. Each data store is connected to a process by means of a data flow symbol. The direction of the data flow arrow shows whether data is being read from or written into a data store.



(5) Output symbol : It is used to represent data acquisition and production during human computer-interaction.

Levels of DFD : There are different levels of data flow diagram. The initial level is called context level or fundamental system model or a 0 level DFD. If two break or expand the 0 level process then we get Ist DFD and if we further expand the Ist level process then we get the 2nd level DFD and so on.

Ans.(ii) Data Dictionary : The data dictionary is an organised listing of all data elements that are pertinent to the system, with precise rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, and components of stores, and even intermediate calculations."

A data dictionary is like any other dictionary, in that it defines each data elements name. The data dictionary contains definitions for all data elements in the system being modelled, no more and no less.

In other words, a data dictionary is a repository (manual or computed-based) containing information about the various data objects appearing on each DFD.

Data dictionary is used to provide an organized approach for representing the characteristics of each data object and control item.

There are no standard formats for the data dictionary. The data store notation in a DFD shows the extistence of one or more data items of stored data.

Generally, data dictionary format may take the following format :

- Name
- Alias
- Where-use/how-used
- Content description
- Supplementary information (type, restrictions or limtation)

Composite data may take the form :

- As a sequence of data items
- As a selection from among a set of data items.
- As a repeated grouping of data items

A data dictionary may just give a simple description of the data. The following is a simple example.

Name = courtesy - title + first -name + (middle-name) + last-name
courtary-title = [Mr. |Miss| Mrs. |Ms.| Dr.| Prof.]
first-name = {legal -character}
last-name = {legal -character}
legal-character = [A-Z | a-z| '| - | |]

Q.5.(a) Give a compiete description about cohesion. (10)

Ans. Cohesion : *"Cohesion is a natural extension of information hiding concept."* Cohesion is a measure of the relative functional strength of a module. The cohesion of a component is a measure of the closeness of the relatioinships between its components. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.

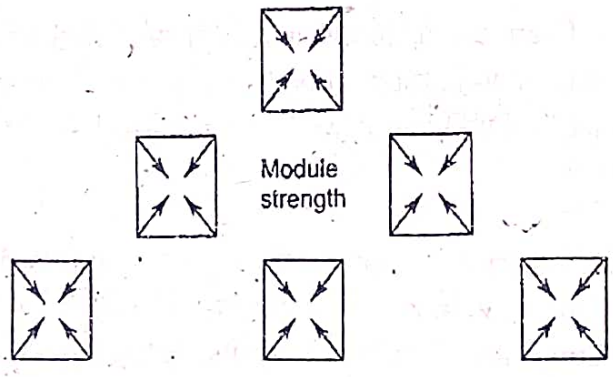


Fig (a) : Cohesion-Strength of Relation within Modules

Types of cohesion : The types of cohesion, in order of the worst to the best type, are as follows :

(i) Coincidental Cohesion : Coincidental cohesion is when parts of a module are grouped arbitrarily (at random); the parts have no significant relationship (e.g. a module of frequently used function).

(ii) Logical Cohesion : Logical cohesion is when parts of a module are grouped because they logically are categorised to do the samething, even if they are different by nature (e.g. grouping all I/O handling routines).

(iii) Temporal Cohesion : Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

(iv) Procedural Cohesion : Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

(v) 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).

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

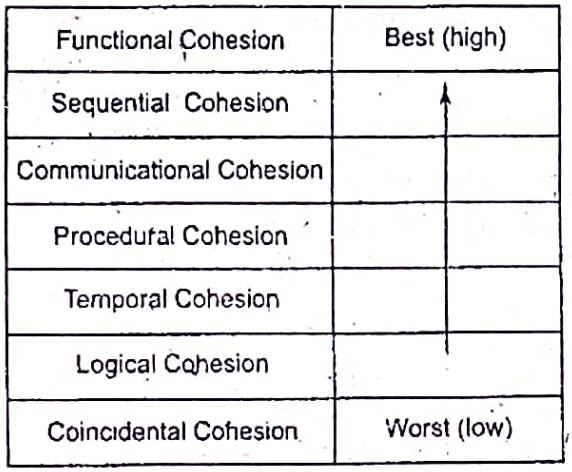


Fig (b) : The Types of Module Cohesion

Best Cohesion: Functional Cohesion
Worst Cohesion: Coincidental Cohesion

Q.5.(b) Explain the following **(10)**

(i) Structural Partitioning

(ii) Control Hierarchy.

Ans. (i) Structural Partitioning : When solving a small problem, the entire problem can be tackled at once. For solving larger problems, the basic principle is the time-tested principle of "divide and conquer." This principle states that divide into smaller pieces, so that each piece can be conquered separately.

For software design, therefore, the goal is to divide the problem into manageably small pieces that can be solved separately. The basic retionale behind this strateggy is the belief that if the pieces of a problem are solvable separately, the cost of solving the entire problem is more than the sum of the cost of solving all the pieces.

However, the different pieces cannot be entirely independent of each other as they together form the system. The different pieces have to co-operate and communicate to solve the larger problem. This communication adds complexity, which arises due to partitioning and may not have existed in the original problem. As the number of components increses, the cost of partitioning, together with the cost of this added complexity, may become more than the saving achieved by partitioning: It is at this point that on further partitioning needs to be done. The designer has to make the judgment about when to stop partitioning.

Structural/Problem partitionary can be divided into two categories :

(a) Horizontal partitioning

(b) Vertical partitioning

(a) Horizontal partitioning : Horizontal partitioning defines separte branches of modular hierarchy for each major program function. The simplest approach to horizontal partitioning defines three partitions-input, data transformation (often called processing) and output Partitioning their architecture horizontally provides a number of distinct benefits :

(i) Software that is easier to test.

(ii) Software that is easier to maintain

(iii) Software that is easier to extend

(iv) Propagation of fewer side effects.

On the negative part, horizontal partitioning often causes more data to be passed across modules interfaces and can complicate the overall control of program flow.

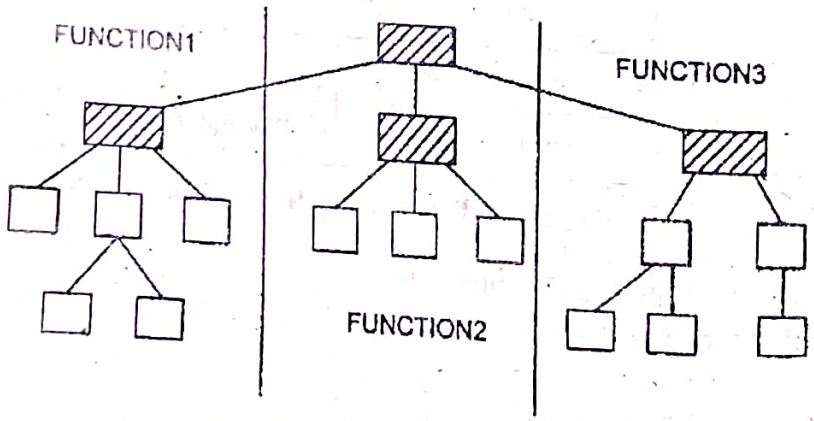


Fig. : Horizontal Partitioning

(b) Vertical partitioning : Vertical partitioning, often called factoring, suggests that control and work should be distributed from top-down in the programme structure. Top level modules should perform control function and do actual processing work. Modules that reside low in the structure should be the workers, performing all input, compilation and output tasks.

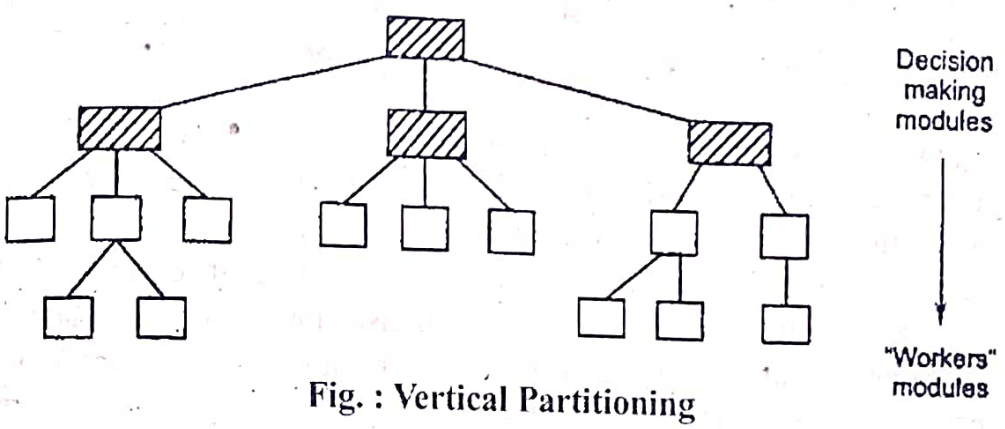


Fig. : Vertical Partitioning

Ans.(ii)Control Hierarchy : Control hierarchy, also called program structure, represents the organization of program components (modules) and implies a hierarchy of control. It does not represent procedural aspects of software such as sequence of processes, occurrence or order of decisions, or repetition of operations; nor is it necessarily applicable to all architectural styles.Different notations are used to represent control hierarchy for those architectural styles that are amenable to this representation. Depth and width provide an indication of the number of levels of control and overall span of control, respectively. Fan-out is a measure of the number of modules that are directly controlled by another module. Fan-in indicates how many modules directly control a given module.The control relationship among modules is expressed in the following way: A module that controls another module is said to be superordinate to it, and conversely, a module controlled by another is said to be subordinate to the controller .The control hierarchy also represents two subtly different characteristics of the software architecture: visibility and connectivity. Visibility indicates the set of program components that may be invoked or used as data by a given component, even when this is accomplished indirectly. For example, a module in an object-oriented system may have access to a wide array of data objects that it has inherited, but makes use of only a small number of these data objects. All of the objects are visible to the module.

Connectivity indicates the set of components that are directly invoked or used as data by a given component. For example, a module that directly causes another module to begin execution is connected to it.

Section - C

Q.6. Write short note on the following: **(20)**

(a) Transform Flow

(b) Transaction Flow

(c) Data Modelling.

Ans. (a) Transform Flow : Information must enter and exit software in an "external world" from. For example data typed on a keyboard, tones on a telephone line, and video images in a multi-media applications are all forms of external world information. Such externalized data must be converted into an internal form for processing. Information enters the system along paths that transform external data into an internal form. These paths are identified as incoming flow. At the kernel of the software, a transition occurs. Incoming data are passed through a transform center and begin to move along paths that now lead "out" of the software. Data moving along these paths are called out going flow.The overall flow of data occurs in a sequential

manner and follows one or only a few, "straight line" paths. When a segment of a data flow diagram exhibits these characteristics, transform flow is present.

Ans.(b)Transaction Flow : Information flow is often characterized by a single data item, called a transaction, that triggers other flow along one of many paths. When a DFD takes the form shown in fig.(1) transaction flow is present

Transaction flow is characterized by data moving along an incoming path that converts external world information into a transaction. The transaction is evaluated and, based on its value, flow along one of many action paths is initiated. The hub of information flow which many action paths emanate is called a transaction center.

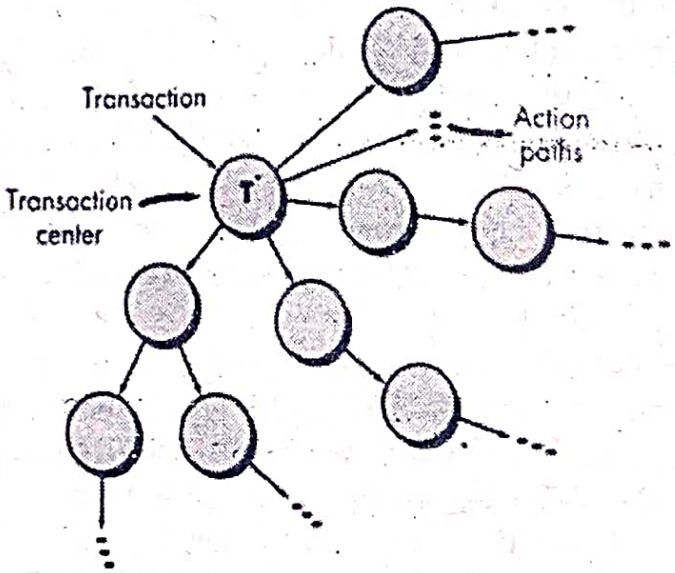


Fig. : (1)

It should be noted that, within a DFD for a large system, both transform and transaction flow may be present. For example, in a transaction-oriented flow, information flow along an action path may have transform flow characteristics.

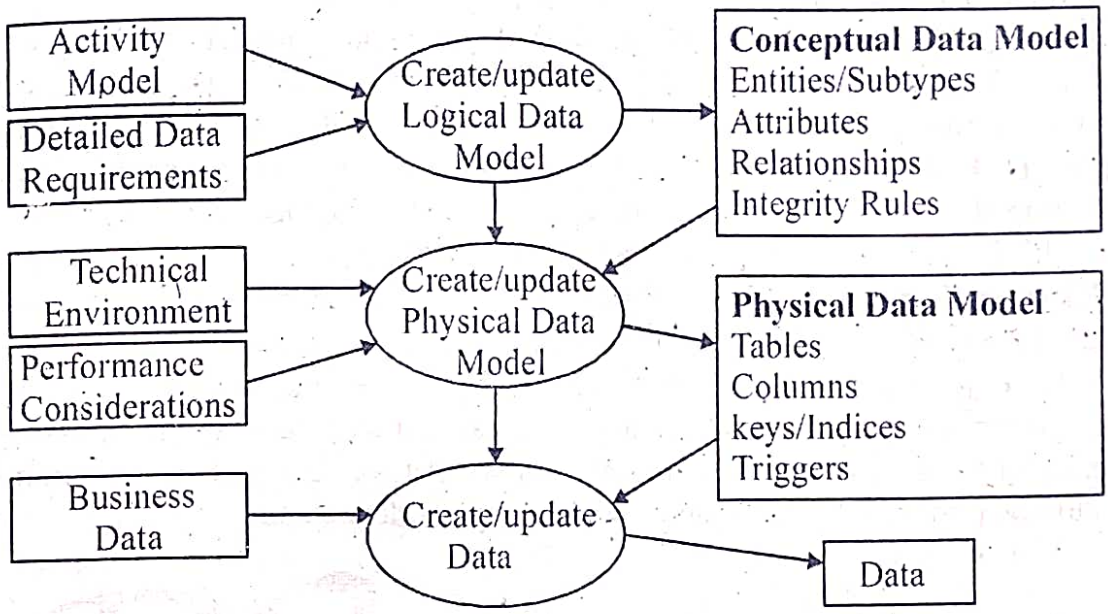
Ans.(c)Data Modelling : Data modeling in software engineering is the process of creating a data model for an information system by applying formal data modeling techniques.

The data modeling process. The figure illustrates the way data models are developed and used today. A conceptual data model is developed based on the data requirements for the application that is being developed, perhaps in the context of an activity model. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects. This is then used as the start point for interface or database design.

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Therefore, the process of data modeling involves professional data modelers working closely with business stakeholders, as well as potential users of the information system.

There are three different types of data models produced while progressing from requirements to the actual database to be used for the information system. The data requirements are initially recorded as a conceptual data model which is essentially a set of technology independent specifications about the data and is used to discuss initial requirements with the business

stakeholders. The conceptual model is then translated into a logical data model, which documents structures of the data that can be implemented in databases. Implementation of one conceptual data model may require multiple logical data models. The last step in data modeling is transforming the logical data model to a physical data model that organizes the data into tables, and accounts for access, performance and storage details. Data modeling defines not just data elements, but also their structures and the relationships between them.



Data modeling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modeling standards is strongly recommended for all projects requiring a standard means of defining and analyzing data within an organization, e.g., using data modeling:

(a) to assist business analysts, programmers, testers, manual writers, IT package selectors, engineers, managers, related organizations and clients to understand and use an agreed semi-formal model the concepts of the organization and how they relate to one another

(b) to manage data as a resource

(c) for the integration of information systems

(d) for designing databases/data warehouses (aka data repositories)

Data modeling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business. The data models should ideally be stored in a repository so that they can be retrieved, expanded, and edited over time. Whitten et al. (2004) determined two types of data modeling:

(i) Strategic data modeling: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for information systems is defined. Information engineering is a methodology that embraces this approach.

(ii) Data modeling during systems analysis: In systems analysis logical data models are created as part of the development of new databases.

Data modeling is also used as a technique for detailing business requirements for specific databases. It is sometimes called database modeling because a data model is eventually implemented in a database.

Q.7.(a) Define software testing. Explain about software Testing strategies. (10)

Ans. Software testing : Software testing is the process of executing a program with the intention of finding errors in the code. It is the process of exercising or evaluating a system or system components by manual or atuomatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

The objects of testing is to show incorrectness and testing is considered to succeed when an error is detected. An error is a conceptual mistake made by either the programmer or the designer or a discrepancy between a computed value and a theoretically correct value. A fault is a specific manifestation of an errror. An error may be the cause of several faults. A failure is the inability of a system or component to perform its required function within the specified limits. A failure may be produced when a fault is executed or exercised.

Testing should not be a distinct phase in system development but should be applicable throughout the design, development and maintenance phases.

Software Testing Strategies : Software testing strategy provides a road map for the software developer, quality assurance organization, and the customer.

A strategy must provide guidance for the practitioner and a set of milestones for the manager. Progress must be measurable and problems must surface as soon as possible.

In order to conduct a proper and through set of tests, the types of testing mentioned below should be performed in the order in which they are described.

(1) Unit Testing
(2) Integration Testing
(3) Functional Testing
(4) Regression Testing
(5) Systems and Acceptance Testing.

However, some system or hardware can happen concurrently, with software testing.

***Unit Testing* :** Unit testing procedure utilizes the white-box method and concentrates on testing individual programming units. These units are somtimes referred to as modules or atomic modules and they represent the smallest programming entity.

Unit testing is essentially a set of path test performed to examine the many different paths through the modules. These types of tests are conducted to prove that all paths in the program are solid and without errors and will not cause abnormal termination of the program or other undesirable results.

***Integration Testing* :** Integration testing focuses on testing multiple modules working together. Two basic types of integration are usually used :

(i) Top-down Integration
(ii) Bottom-up Integration

Above types of integration are discussed below :

Top down Integration : As the term suggest, starts at the top of the program hierarchy and travels down its branches. This can be done in either depth-first (shortest path down to the deepest level) or breadth-first (across the hierarchy, before proceeding to the next level).

The main advantage of this type of integration is that the basic skeleton of the program/ system can be seen and tested early.

The main disadvantage is the use of program stubs until the actual are written. This basically limits the up-flow of information and therefore does not provide for a good test of the top-level modules.

Bottom-up Integration : This type of integration has the lowest level modules built and tested before its utilized by its calling module.

This method has a great advantage in unconvering errors in critical modules early.

Main disadvantage is the fact that most or many modules must be build before a working program can be presented.

Functional Testing **:** Functional testing verifies that an application does what it is supposed to do and doesnot do what it shouldn't do.

For example, if you are functionally testing a word processing application, a partial list of checks you would perform minimally includes creating, saving, editing, spell checking and printing documents.

Function testing usually includes testing of all the interfaces and should therefore involve the clients in the process. Because every aspect of the software system is being and how will conduct the tests and what exacutly will be tested.

Regression Testing : Regression testing is the process of running a subset of previously executed integration and function tests to ensure that program changes have not degraded the system. The regressive phase concerns the effects of newly introduced changes on all the previously integrated code. Problems arise when errors made in incorporating new functions affects the previously tested functions, which are common in large systems.

Regression testing may be conducted manually or using automated tools. The basic regression resting approach is to incorporate selected test cases into a regression bucket that is run periodically to find regression problems. In many organizations regression testing consists of running all the functional tests every few months. This generally delays the regression problem detection and results in significant rework after every regression run.

It is wise to accumalte a comprehensive regression bucket and also define a subset of test cases. The full bucket in run occasionally, but the subset is run against every spin. The spin subset consist of all test cases for recently integrated functions and a selected sample from the full regression bucket. Depending on the success history of test subsets and the complexity of the program the test cases are selected from the subset.

Systems and Acceptance Testing **:** Final stage of the testing process should be system Testing and Acceptance Testing. It is concerned with the execution of test cases to evaluate the whole system with respect to the user's requirement. A system test checks for unexpected interaction between the units and modules, and also evaluates the system for compliance with functional requirements.

An acceptance test is the process of executing the test cases agreed with the customer as being an adequate representation of user requirments. These are often called Black Box or Functional tests.

Q.7.(b) Explain in detail about the term Debugging. (10)

Ans. Debugging : Debugging means identifying, locating and correcting the bugs usually by running the program. It is an extensively used term in programming. These bugs are usually logical errors.

During compilation phase the source files is accessed and if errors are found, then that file is edited and the corrections are posted in the file. After the errors have been detected and the corrections have been included in the source file, the file is recompiled. This detection of

errors and removal of those errors is called debugging. The file is compiled again, so changes done last time gets included in the objects file also by itself. This process of compilation, dedugging and correction posting in the source file contiunes until all syntactical errors are removed completely. If program is very large and complex, more is the number of times the program has to be corrected and compiled.

Successful compilation of the program means that now the program is following all the rules of the language and ready to execute. All of the syntax errrors of the program are indicated by the compiler at this stage.

Debugging Tactics/Categories : The various catagories for debugging are

(i) Brute force debugging
(ii) Backtracking
(iii) Cause elimination
(iv) Program slicing
(v) Fault tree analysis.

(i) Brute Force Debugging : The programmer appends the print or write statements which, when executed, display the value of a variable. The programmer may trace the value printed and locate the statements containing the error. Earlier when the time for execution was quite high, programmers had to use the core dumps. The core dumps are referred to as the static image of the memory and this may be scanned to identify the bug.

(ii) Backtracking : In this technique, the programmer back tracks from the place or statement which gives the errors symptom for the first time. From this place, all the statements are checked for possible causes of errors. Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.

(iii) Cause Elimination : Cause elimination is mainfested by induction or deduction and introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.

A list of all possible causes is developed and tests are conducted to eliminate each. If initial tests indicate that a particular cause hypothesis shows promise ; the data are refined in an attempt to isolate the bug.

(iv) Program slicing : This tecnique is similar to backtracking. However, the search space is reduced by defining slices. A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable.

(v) Fault-Tree Analysis : Fault-tree analysis, a method originally developed for the U.S Minute man missible program, reasons about the design helping us to decompose it and look for situations that might lead to failure. In this sense, the name is misleading ; we are really analyzing failures, not faults, and looking for potential causes of those failure. We built fault trees that display the logical path from effect to cause. These trees are then used to support fault correction or tolerance, depending on the design strategy we have chosen.

Debugging process : Debuggging is not testing but always occurs as a consequence to testing. Referring to Fig.(1), the debugging process begins with the execution of a test case. Results are assessed and a lack of correspondence between expected and actual performance is encountered. In many cases, the no corresponding data are a symptom of an underlying cause

as yet hidden. Debugging attempts to a match symptom with cause, thereby leading to error correction.

Debugging will always have one to two outcomes :

(i) The cause will be found and corrected and removed.

(ii) The cause will not be found.

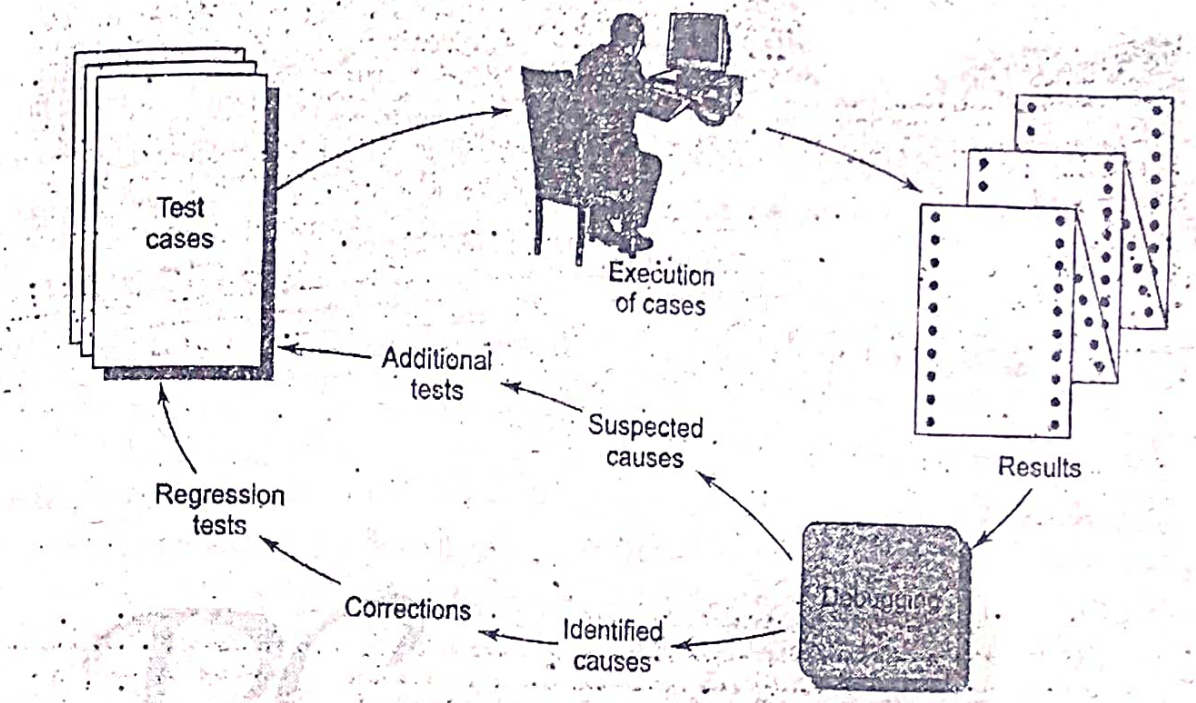


Fig. : (1)

Section - D

Q.8. Describe the following : (20)

(a) Software configuration Management.

(b) Software Quality Assurance.

(c) Software Reliability.

Ans. (a) Software configuration Management : Software configuration management (SCM) is one of the foundations of software engineering. It is used to track and manage the emerging product and its versions. This is to assure quality of the product during development, and operational maintenance of the product. SCM ensures that all people involved in the software process know what is being designed, developed, built, tested, and delivered.

Definitions : Different people perceive software Configuration Management differently. The following are few perceptions of several people about SCM in the form of definitions.

(i) Software Configuration Management (SCM) is the art of identifying organizing and controlling modifications to the software being built by a programming team. The goal of is to maximize productivity by minimizing mistakes.

(ii) Software Configuration Management (SCM) is the process of defining and implementing a standard configuration, that results into the primary benefits such as easier set-up and maintenance, less down time, better integration with enterprise management, and more efficient and reliable backups.

(iii) Software Configuration Management (SCM) is the process concerned with the development of procedures and standards for managing an evolving software system product.

(iv) Software Configuration Management is the ability to control and manage change in a software project.

Importance of SCM : SCM gives developers a means to coordinate the work of numerous people on a common product whether they work in close proximity or over a wide geographical area. Without a good SCM plan, projects become increasingly difficult to control and can reach the point where the project has to be discontinued because it can not be fixed. There is an impact of software lifecycle model on software configuration management.

Goals : The following are the major goals of software configuration management :

(i) Software configuration management activities are planned.

(ii) Selected software work products are identified, controlled and available.

(iii) Change to identified software work products are controlled.

(iv) Affected groups and individuals are informed of the status and content of software baseline.

Ans.(b) Software Quality Assurance : The aim of the software Quality Assurance (SQA) process is to develop high-quality software product. Software Quality Assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained.

Quality assurance is a planned and systematic pattern of all actions necessary to provide adequate confindence that the item or product conforms to established requirements.

The purpose of a software quality assurance group is to provide assurance that the procedures tools, and techniques used during product development and modification are adequate to provide the desired level of confidence in the work products.

The process of software Quality Assurance : (i) Defines the requirements for software controlled system fault/failure detection, isolation, and recovery.

(ii) Reviews the software developement process and products for software error prevention and/or controlled change to reduced functionality states.

(iii) Defines the process for measuring and analyzing defects as well as reliability and main tainability factors.

SQA Objectives : The various objectives of SQA are as follows :

(i) Quality management approach.

(ii) Measurement and reporting mechanisms.

(iii) Effective software engineering technology.

(iv) A procedure to assure compliane with software development standards where applicable.

(v) A multi-testng strategy is drawn.

(vi) Formal technical reviews that are applied throughout the software process.

SQA Goals : The major goals of SQA are as follows :

(i) SQA activities are planned.

(ii) Non-compliance issues that cannot be resolved within the software project are addressed by senior management.

(iii) Adherence of software products and activities to the applicable standards, procedures and requirements is verified objectively.

(iv) Affected groups and individuals are informed of SQA activities and results.

SQA Plan : A plan that defines the quality process and procedures that should be used. This involve selecting and instantiating standards for products and process and defining the required quality attributes of the system.

The SQA Plan provides a road map for instituting software quality assurance. Developed by SQA group (or the software team if a SQA group does not exist), the plan serves as a template for SQA activities that are instituted for each software project.

The quality plan should select those organizational standards that are appropriate to a particular product and developement process. New standards may have to be defined if the project uses new methods and tools.

An outline structure for a quality plan includes :

(i) *Product intoduction* : A description of the product, its intended market and the quality expectations for the product.

(ii) *Product plans* : The critical release dates and responsibilities for the product along with plans for distribution and product servicing.

(iii) *Process descriptions* : The development and service processes that should be used for product development and management.

(iv) *Quality goals* : The quality goals and plans for the product including an identification and justification of crictical product quality attributes.

(v) *Risks and risk management* : The key risks that might affect product quality and the actions to address these risks.

Ans.(c) Software Reliability : "Software Reliability is defined as the probabilty of failure-free software operation for a specified period of time in a specified environment".

Informally, Software Reliability can be defined as a measure of how well the users think the system operators according to its specifications".

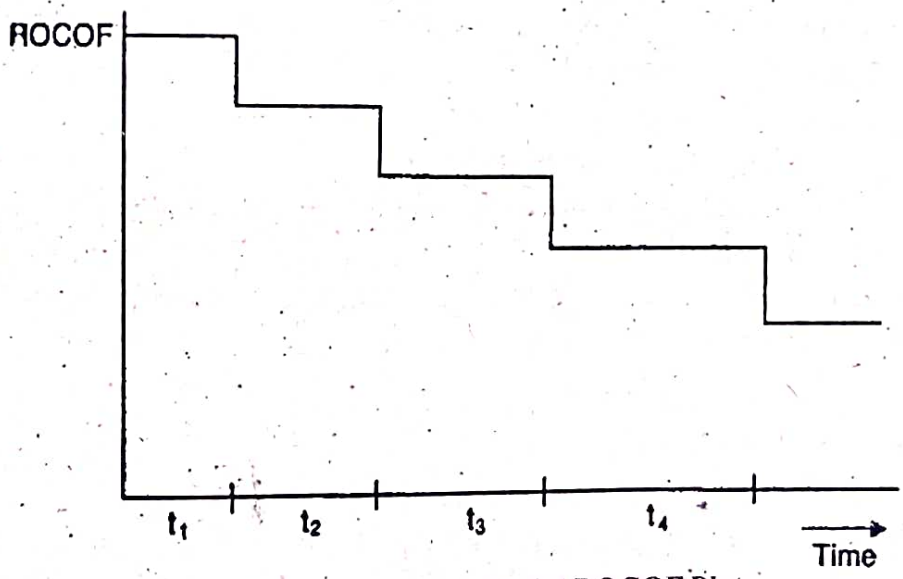


Fig : Jelinksi-Moranda Model ROCOF Plot

Jelinski-Moranda Model : The JM model is the best-known model of reliability models. This model is based in Markov process. The Jelinski-Moranda model says that the hazard

rate is a step function, where improvements in reliability only takes place when a failure is fixed, and each failure contributes equally to the overall reliability. The plot of Rate Of Occurrence Of Falilures (ROCOF) versus time for the Jelinski-Moranda Model is shown in Fig.

The ROCOF is defined as the probability of a failure, not necessarily the first, in a defined interval.

It assumes that for each i,

$$F_i(t_i) = 1 - e^{-\lambda_i t_i} \text{ with } \lambda_i = (N - i + 1)\phi$$

Here N is the initial number of faults, and f is the contribution of each fault to the overall failure. The model also assumes that all faults have the same rate. The inference procedure of JM is called maximum likelihood estimation. For a particular set of failure data, this procedure produces etimates of N_i and ϕ_i. Then t_i is predicted by putting these estimates into the model.

Following are two important reliability models based on JM model :

(i) *Shooman's Reliability Model*

(ii) *Musa's Reliability Model*

These models are briefly discussed below.

Shooman's Reliability Model : Various reliability models based on JM have been produced. Shooman's model is also a similar model with some additional assumptions and postulates, such as :

– Hazard rate is proportional to the number of remaining errors.

– Hazards are detemined by the rate at which the remaining errors were passed in the execution of the program.

Thus, hazard rate depends on depends on the instruction processing rate, the size of the program and the number of errors remaining in the program.

Musa's Reliability Model : The Musa reliability model has JM model as its basis but builds features into it (Musa 1987). It is the first model to use execution time to gain inter-failure times. Musa decided that software reliability theory should be based on execution time rather than calendar time.

The execution time model is richer in simplicity and clarity of modelling and has better conceptual insight and predictive validity.

Musa model views execution time in two ways :

– Operating time of the software product and

– Cumulative execution time that occurs during the test phase fo development and the post delivery maintenance.

The hazard rate was assumed to be constant w.r.t. operating time, but it varies as a function of the errors remaining and hence cumulative execution time.

Q.9. Give a complete description about computer Aided software Engineering. (20)

Ans. CASE stands for computer aided software engineering.

CASE is a tool which aids a software engineer to maintain and develop software. The workshop for software engineering is called in Integrated Project Support Environment (IPSE) and the tool set that fills the workshop is called CASE.

CASE is a computer aided software engineering technology CASE is an automated support tool for the software engineers in any software engineering process.

Software engineering mainly includes the following processes :

1. Translation of user needs into software requirements.
2. Transaction of software requirements into design specification
3. Implementation of design into code
4. Testing of the code
5. Documentation

CASE technology provides software process support by automating some process activities and by providing information about the software, which is being developed. Examples of activities, which can be automated using CASE, include:

1. The development of graphical system models as part of the requirements specification or the software design.
2. Understanding a design using a data dictionary, which holds information about the entities and relations in a design.
3. The generation of user interfaces from a graphical interface description, which is created interactively by the user.
4. Program debugging through the provision of information about an executing program.
5. The automated translation of programs from an old version of a programming language such as COBOL to a more recent version.

The use of Computer Aided Software Engneering (CASE) tool reduce the effort of development of achieving quality goals and managing change and configuration throughout the product life cycle. It also help the project manager, the software developer and other key personnel to improve their productivity in the development team.

CASE Tools : CASE Tools are software programs that are designed to assist human programmers with the complexity of the processes and the artifacts of software engineering. They constitute the laws of and the automated tools that aid in the synthesis, analysis, modelling, or documentation of software.

The schematic diagram of CASE tools is shown in fig.

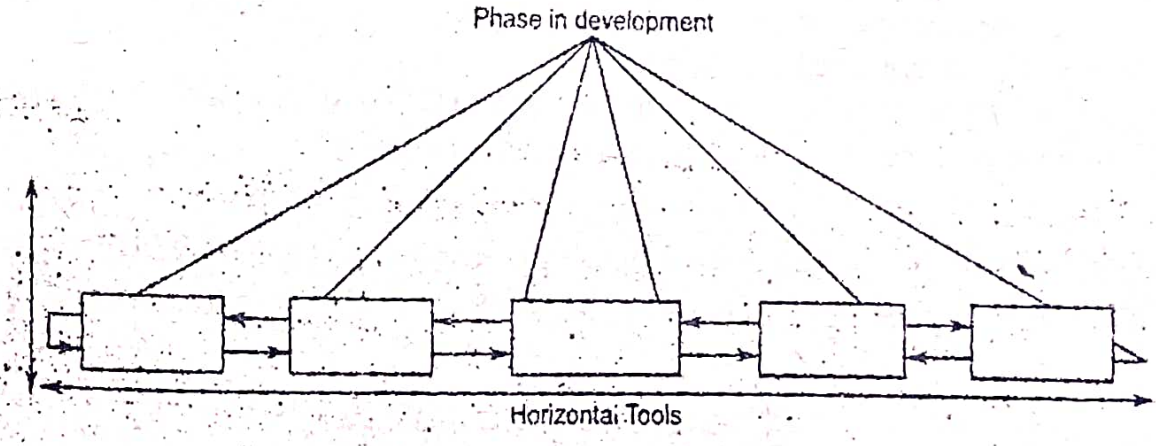


Fig. : Categories of case tools

Smith and Oman have defined CASE tools which are divided into the following two categories :

(i) Vertical CASE tools
(ii) Horizontal CASE tools

(i) Vertical CASE Tools : Vertical CASE tools provide support for certain activities within a single phase of the software life cycle.

There are two subcategories of vertical CASE tools :

– *First category :* It is the set of tools that are within one phase of life cycle. These tools are important so that development in each phase can be as quick as possible.

– *Second category :* It is a tool that is used in more than one phase, but does not support moving from one phase to the next. These tools ensure that the developer does move on the next phase as appropriate.

(ii) Horizontal CASE tools : These tools support automated transfer of information between the phases of a life cycle. These tools include project management, configuration management tools and integration services.

Various **advantages of CASE tools** are as follows :

(i) Improved productivity
(ii) Better documentation
(iii) Improved accuracy
(iv) Intangible benefits
(v) Improved quality
(vi) Reduced lifetime maintenance
(vii) Opportunity to non-programmers
(viii) Reduced cost of software
(ix) Produce high quality and consistent documents
(x) Impact on the style of a working of company
(xi) Reduce the drudgery in a software engineer's work
(xi) Increase speed of processing
(xii) Easy to program software

Various **disdvantages of CASE tools** are as follows :

(i) *Purchasing of case tools is not and easy task :* Its cost is very high. Due to this reason small software development firm do not invest in case tools.

(ii) *Learning curve :* In general cases programmer productivity may fall in initial phase of implementation as user need time to learn this technology.

(iii) *Tool mix :* It is important to make proper selection of case tools to get maximum benefit from the case tool, so wrong selection may lead to wrong result.

PRINCIPLES OF SOFTWARE ENGINEERING

May - 2018

Paper Code:-CSE-302-F

Note : *Attempt five questions in all, selecting one question from each Section. Question No. 1 is compulsory. All questions carry equal marks.*

Q.1. Write the short note on the following :

(a) Process

(b) Software Engineering

(c) Data Dictionary

(d) Abstraction

(e) Stress Testing

(f) Recovery Testing

(g) The ISO 9001 standard

(h) Repository

Ans.(a) Process : *A process is a series of steps involving activities, constraints and resources that produce an intended output of some kind.*

A process has the following characteristics :

(i) The process prescribes all of the major process activities.

(ii) The process uses resources, subject to a set of constraints (such as a schedule), and produces intermediate and final products.

(iii) The process may be composed of sub processes that are linked in some way. The process may be defined as a hierarchy of processes, organized so that each sub process has its own process model.

Ans.(b) Software Engineering : Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software.

According to Barry Boehm : Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation.

Ans.(c) Data Dicitionary : "The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of input, outputs, and components of stores, and even intermediate calculations."

A data dictionary is like any other dictionary, in that it defines each data element name. The data dictionary contains definitions for all data elements in the system being modelled, no more and no less.

Ans.(d) Abstraction : An abstraction of a component describes the external behavior of that component without bothering with the internal details that produce the behavior.

Abstraction is an indispensable part of the design process and it is essential for problem partitioning. Partitioning essentially is the exercise in determining the components of a system. However, these components are not isolated from each other, but interacts with other components. In order to allow the designer to concentrate on one component at a time, abstraction of other component is used.

Abstraction is used for existing components as well as component that are being designed. Abstraction of existing components plays an important role in the maintenance phase.

Ans.(e) Stress Testing : Stress testing refers to the testing of software or hardware to determine whether its performance is satisfactory under any extreme and unfavorable conditions, which may occur as a result of heavy network traffic, process loading, underclocking, overclocking and maximum requests for resource utilization.

Most systems are developed under the assumption of normal operating conditions. Thus, even if a limit is crossed, errors are negligible if the system undergoes stress testing during development.

Ans.(f) Recovery Testing : Recovery testing uses test cases designed to examine how easily and completely the system can recover from a disaster (power shut down, blown circuit, disk crash, interface failure, insufficient memory, etc). It is desirable to have a system capable of recovering quickly and with minimal human intervention. It should also have a log of activities happening before the crash (these should be part of daily operations) and a log of messages during the failure (if possible) and re-start.

Ans.(g) ISO 9001 standard : ISO 9001 is a Quanlity assurance standard that is specific to software engineering. It specifies 20 standards with which an organization must comply for an effective implementaion of the quality assurance system. It is an international quality management syatem. ISO 9001 is mainly related to the software industry. It lay down the standards for designing, developing, servicing the producing of standard quality of goods. It is also applicable to most software development organizations.

The salient features of ISO 9001 requirements are as follows :

(i) All documents concerned with the development of a software product should be properly managed, authorized and controlled.

(ii) Proper plans should be prepared and then progress against these plans should be monitored.

(iii) Important documents should be independently checked and reviewed for effectiveness and correctness.

(iv) The product should be tested against its specification.

Ans.(h) Repository : In software development, a repository is a central file storage location. It is used by version control systems to store multiple versions of files. While a repository can be configured on a local machine for a single user, it is often stored on a server, which can be accessed by multiple users.

A repository contains three primary elements — a trunk, branches, and tags. The trunk contains the current version of a software project. This may include multiple source code files, as well as other resources used by the program. Branches are used to store new versions of the program. A developer may create a new branch whenever he makes substantial revisions to the

program. If a branch contains unwanted changes, it can be discontinued. Otherwise, it can be merged back into the trunk as the latest version. Tags are used to save versions of a project, but are not meant for active development. For example, a developer may create a "release tag" each time a new version of the software is released.

Unit – I

Q.2. What is Software Life Cycle Model ? Explain its types with their advantages and limitation.

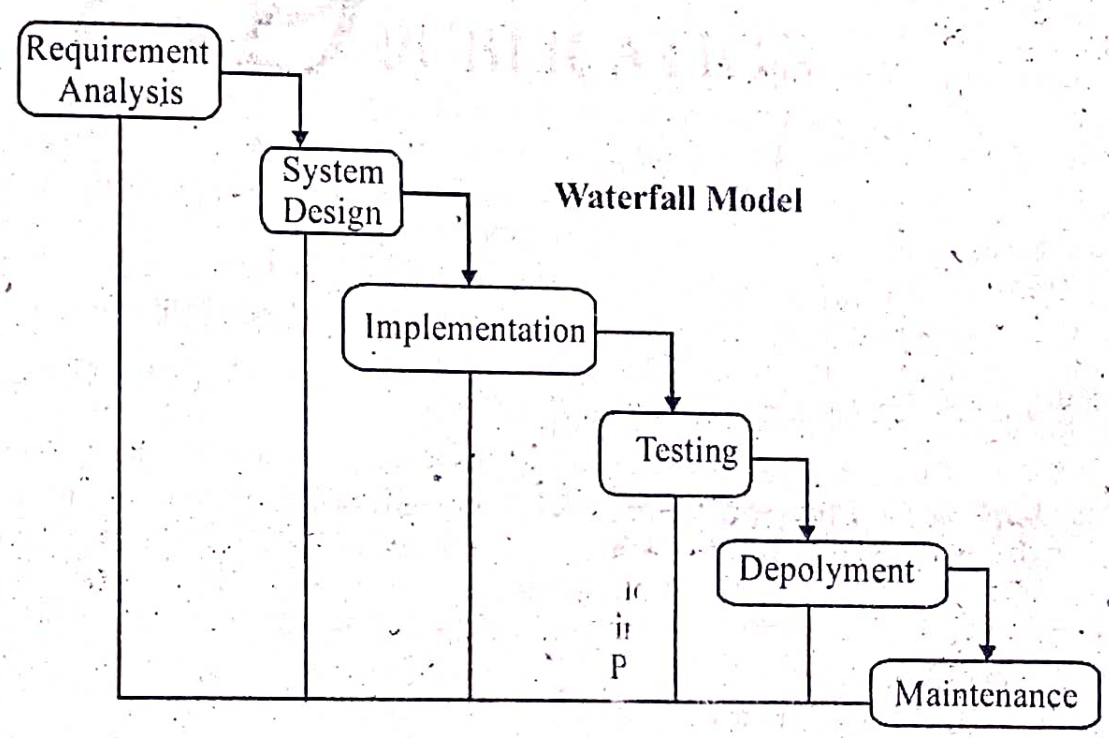
Ans. There are various software development life cycle (SDLC) models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development.

Following are the most important and popular SDLC models followed in the industry:

1. Waterfall Model
2. Iterative Model
3. Spiral Model
4. V-Model
5. Big Bang Model

The other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

1. SDLC Waterfall Model : Following is a diagrammatic representation of different phases of waterfall model.



Waterfall Model

The sequential phases in Waterfall model are:

Requirement Gathering and analysis : All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

System Design: The requirement specifications from first phase are studied in this phase

and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

Implementation : With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

Integration and Testing : All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system : Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.

Maintenance : There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

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

Advantages of Water fall Model :

(i) It is a linear model.
(ii) It is a segmental model.
(iii) It is systematic and sequential.
(iv) It is simple one.
(v) It has proper documentations.

Disadvantages of Waterfall Model :

(i) It is difficult to define all requirements at the beginning of a project.
(ii) This model is not suitable for accommodating any charge.
(iii) A working version of the system is not seen until late in the project's life.
(iv) It does not scale up well to large project.
(v) It involves heavy documentation.
(vi) We cannot go in the backward direction while SDLC performs.
(vii) There is no sample model for clearly in realization the customers needs.
(viii) There is no risk analysis.
(ix) If there is any mistake or error in any phase then we cannot make good software.
(x) It is a document driven process that requires formal documents at the end of each phase.

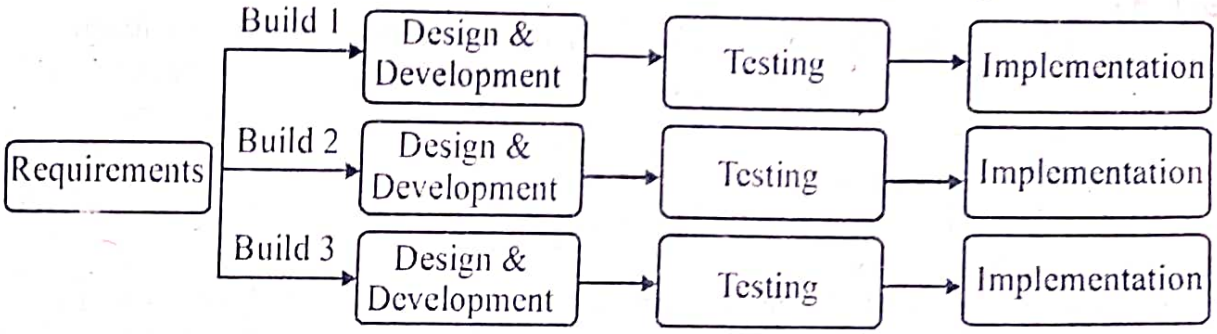
2. SDLC Iterative Model : Following is the pictorial representation of Iterative and Incremental model:

This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team

while working on the project.

- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.



Advantages of Iterative Enhancement Model : The various advantages of following the approach of the iterative enhancement are as follows :

(i) The feedbacks from early increments improve the later stages.

(ii) The possibility of changes in requirement is reduced because of the shorter time span between the design of a component and its delivery.

(iii) Users get benefits earlier than with a conventional approach.

(iv) Early delivery of some useful components improves cash flow, because you get some return on investment early on.

(v) Smaller sub-projects are easier to control and manage.

(vi) 'Gold-plating', that is the requesting of features that are unnecessary and not in fact used, is less as users will know that they get more than one bite of the cherry if a feature is not in the current increment then it can be included in the next.

(vii) The project can be temporarily abandoned if more urgent work crops up.

(viii) Job satisfaction is increased for developers who see their labours bearing fruit at regular, short, intervals.

Disadvantages of Iterative Enhancement Model : The various disadvantages of iterative enhancement model have been put forward :

(i) Software breakage, that is, later increments may require modifications to earlier increments.

(ii) Programmers may be more productive working on one large system than on a series of smaller ones.

(iii) Some problems are difficult to divide into functional units (modules), which can be incrementally developed and delivered.

(iv) Testing of modules result into overhead and increased cost.

3. SDLC Spiral Model : The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification : This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in

the identified market.

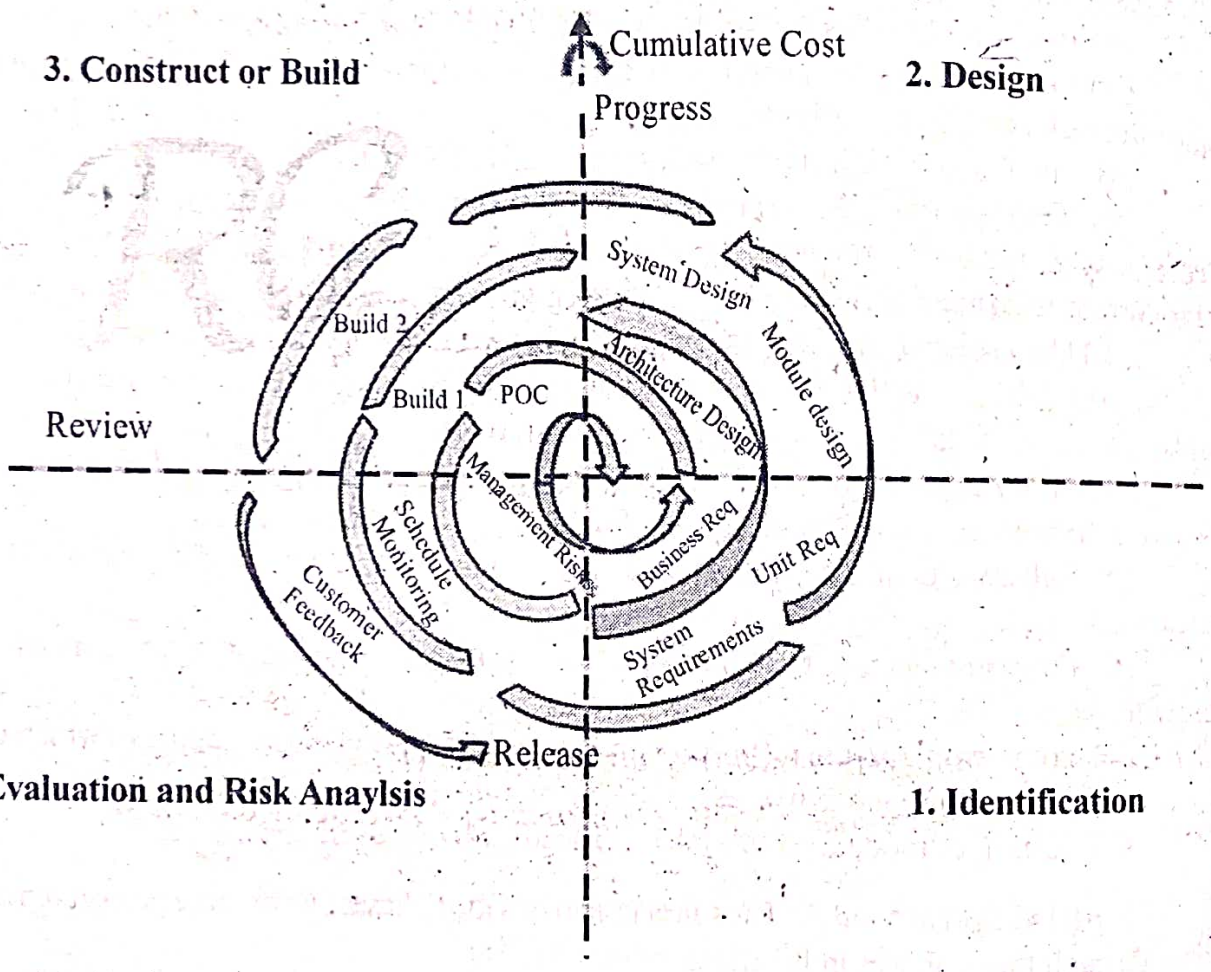
Design : Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct or Build : Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Evaluation and Risk Analysis : Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Following is a diagrammatic representation of spiral model listing the activities in each phase:



Limitations of Spiral Model :

(i) No strict standards for software development.

(ii) No particular beginning or end of particular phase.

Advantages of Spiral Model :

(i) It is risk driven model.

(ii) It is very flexible.

(iii) Less documentation is needed.

(iv) It uses prototyping.

(v) It is more realistic model for software development.

Disadvantages of Spiral Model : The spiral model has some disadvantages that need to be resolved before it can be a universally applied life cycle model. The disadvantages include lack of explicit process guidance in determining objectives, constraints, alternatives, relying on risk assessment expertise and providing more flexibility than required for many applications.

Other disadvantages are :

(i) Model is not suitable for small project as cost of risk analysis may exceed the actual cost of the project.

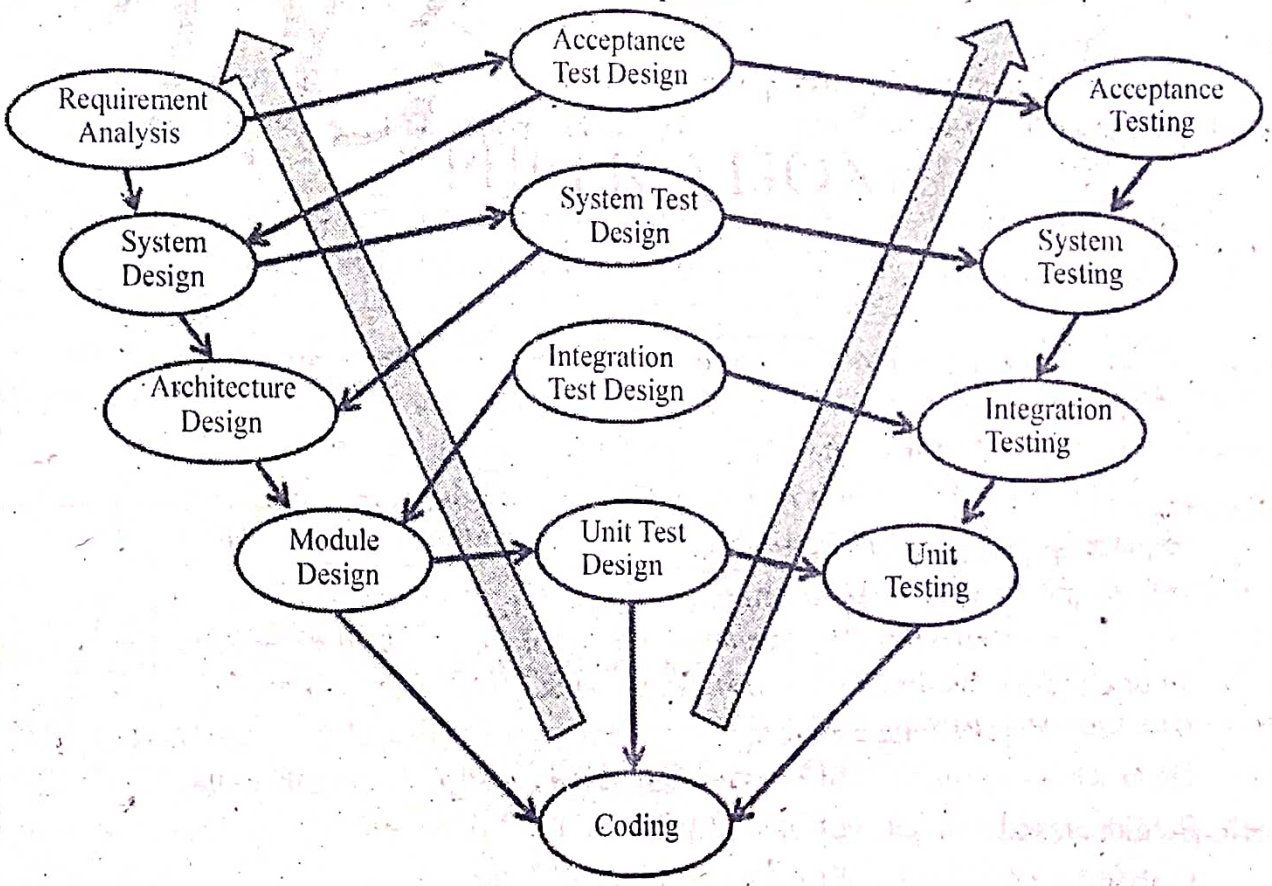
(ii) Model requires expertise in risk management and excellent management skills.

(iii) Different persons involved in the project may find it complex to use.

4. V Model : The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

The below figure illustrates the different phases in V-Model of SDLC.



5. SDLC Big Bang Model : The Big Bang model is SDLC model where there is no specific process followed. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer

requirement.

Big Bang Model is SDLC model where there is no formal development followed and very little planning is required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

Q.3. Explain COCOMO-A Heuristic estimation techniques with its variants.

Ans. COCOMO model : COCOMO stands for *Constructive Cost Model*. It was introduced by Barry Boehm in 1981. It is perhaps the best known and most throughly documented of all software cost estimation models. It provides the following three levels of models :

(i) *Basic COCOMO model :* The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions :

$$\text{Effort} = a_1 \times (KLOC)^{a_2} \quad PM$$

$$T_{dev} = b_1 \times (Effort)^{b_2} \quad Months$$

Where, *KLOC* is the estimated size of the software product expressed in kilo lines, Code a_1, a_2, b_1, b_2 are constants of software products.

T_{dev} is the estimated time to develop the software, expressed in months.

Effort is the total effort required to develop the software product, expressed in person-month (PM).

Person months curve is shown in fig.(1).

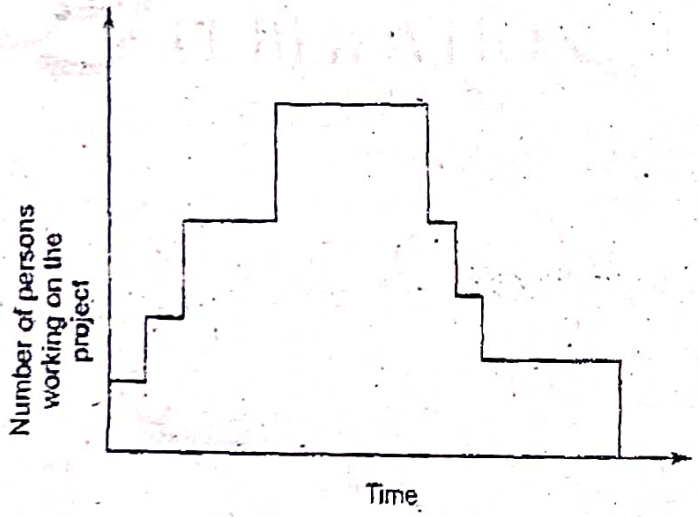


Fig.(1) : Person-month curve

Estimation of Development effort :

Organic: Effort = $2.4(\text{KLOC})^{1.05}$ PM

Semidetached : Effort = $3.0(\text{KLOC})^{1.12}$ PM

Embedded : Effort = $3.6(\text{KLOC})^{1.20}$ PM

Fig.2(a) shows a plot of estimated effort versus size for various product sizes.

From fig.2(a), we can observe that the effort is almost linearly proportional to the size of the software product.

Estimation of Development time :

Organic: $T_{dev} = 2.5(\text{Effort})^{0.38}$ Months

Semidetached: $T_{dev} = 2.5(\text{Effort})^{0.35}$ Months

Embedded: $T_{dev} = 2.5(\text{Effort})^{0.32}$ Months

Fig.2(b) shows the plot is development time versus product size.

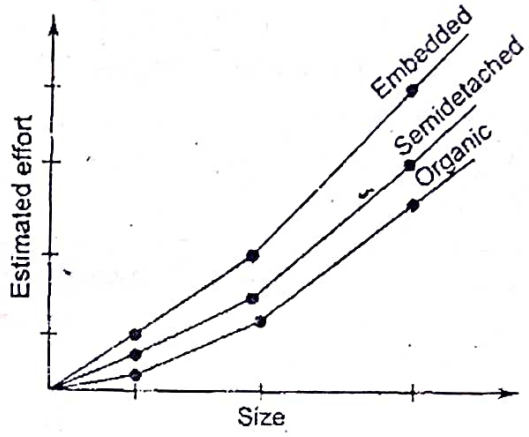


Fig.2(a) : Effort versus size

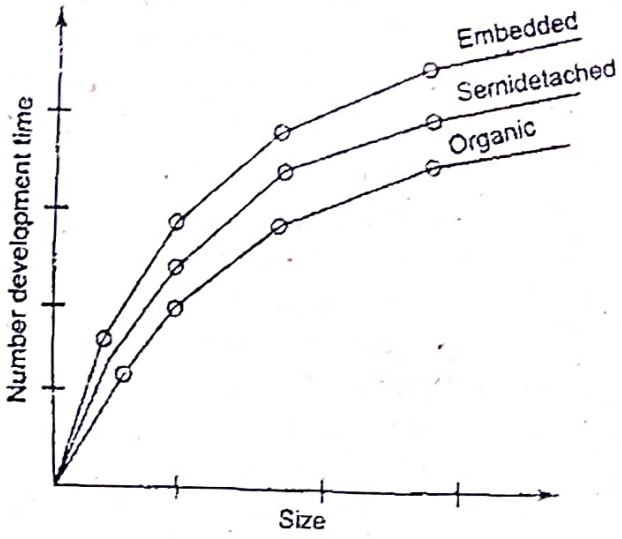


Fig.2(b) : Development Time versus size

From fig.2(b), we can observe that the development time is sub linear function of the size of the product.

(ii) *Intermediate COCOMO Model :* The intermediate COCOMO model recognizes this fact and refines the intial estimate obtained through the BASIC COCOMO expression by using a set of 15 cost drivers (multipliers) based on various attributes of software development.

Table : COCOMO Intermediate Cost drivers

Driver type	Code	Cost Driver
Product attributes	RELY	Required software reliability
	DATA	Database size
	CPLX	Product complexity
Computer attributes	TIME	Execution time constraints
	STOR	Main storage constraints
	VIRT	Virtual machine volatility-degree to which the operating system changes
	TURN	Computer turn around time
Personnel attributes	ACAP	Analyst capability
	AEXP	Application experience
	PCAP	Programmer capability
	VEXP	Virtual machine (i.e.,operation system) experience
	LEXP	Programming language experience

Project attributes	MODP	Use of modern programming practices
	TOOL	Use of software tools
	SCED	Required development schedule.

(iii) *Complete COCOMO Model :* The short-comings of both basic and intermediate COCOMO models are that they :

– Consider a software product as a single homogeneous entity.

– However, most large systems are made up of several smaller sub-systems. Some sub-systems may be considered as organic type, some embedded, etc. For some the reliability requirements may be high and so on.

– Cost of each sub-system is estimated separately.

– Costs of the sub-systems are added to obtain total-cost.

– Reduces the margin or error in the final estimate.

A Large amount of work has been done by Boehm to capture all significant aspects of a software development. It offers a means for processing all the project characteristics to construct a software estimate.

Unit – II

Q.4. Give different Design Principles in detail with suitable examples.

Ans. Design principles : Principle in designing a software are as follows :

(i) Abstraction : Abstraction (separates concepts from instantiations). It allows designers to focus on solving a problem without being concerned about irrelevant lower level details.

– Procedural abstraction - named sequence of events.

– Data abstraction - named collection of data objects.

(ii) Refinement : It's the process of elaboration where the designer provides successively more detail for each design component.

– Decompose design decisioin top – down to elementary level.

– Isolate design aspects that are not independent.

– Add details incrementally each step.

– Postpone decisions relating to detailed representations.

– Continually demonstrate that each refinement step is a correct expansion of previous steps.

(iii) Modularity : A module is a named entity that :

– Contains instructions, processing logic, and data structures.

– Can be separately compiled and stored in a library.

– Can be included in a program.

– Module segments can be used by invoking a name and some parameters.

– Modules can use other modules.

Modularity is the degree to which software can be understood by examining its components independently of one another.

(iv) Software Architecture : Overall structure of the software components and the ways in which that structure provides conceptual integrity for a system.

Software architecture is a sketchy map of the system. Software architecture describes the coarse grain components (usually descibes the computation) of the system. The connectors between these components describe the communication, which are explicit and pictured in a relatively detailed way. In the implementation phase, the coarse components are refined into "actual components", e.g, classes, and Objects. In the object-oriented field, the connectors are usually implemented as interfaces, Control hierarchy or program structure.

(v) Structural Planning : In *horizontal paritioning* the control modules are used to co-ordinate communication between and execution of the functions. Partitioning this way provides the following benefits :

(i) Results in software that is easier to test and maintain.

(ii) Results in fewer propagation side-effects.

(iii) Results in software that is easier to extend.

In *vertical partitioning* the control (decision making) modules are located at the top, and work is distributed top-down in the program structure. That is, top level functions perform control functions and do little actual processing, while modules that are low in the structure perform all input, computation and output tasks. As changes to programs usually revolves around one of these three tasks there is less likelihood that changes made to the lower modules will propagate (upwards) making this partitioning strategy more maintainable.

(vi) Data Structure : It's the representation of the logical relationship among individual data elements. (requires at least as such attention as algorithm design).

Software Procedure : The precise specification of processing :

- Event sequences.
- Decision points.
- Repetitive operations.
- Data organization/structure.

(vii) Information Hiding : The principle of information hiding is the hiding of design decision in a computer program that are most likely to change, thus protecting other parts of the program from change if the design decision is changed. Information (data and procedure) contained within a module is inaccessable to modules that have no need for such information. It should be used to guide architectural design, interface designs, and modularization. Modules hide difficult or changeable design decisions.

Q.5. What is Functional Independence explain in detail ?

Ans. Functional Independence : Functional independence is the refined form of the design concepts of modularity, abstraction and information hiding. Functional independence is achieved by developing a module in such a way that it uniquely performs a given set of functions without interacting with order parts of the system. The software that uses the property of functional independence is easier to develop because its functions can be categorized in a systematic manner. Moreover, independent modules require less maintenance and testing activity as the secondary effects caused by design modification are limited due to less propagation of errors. In short, it can be concluded that functional independence is the key to a good software design and a good design results in high-quality software. There exist two qualitative criteria for measuring functional independence, namely, coupling and cohesion.

Cohesion : *"Cohesion is a natural extension of information hiding concept."* Cohesion is a measure of the relative functional strength of a module. The cohesion of a component is a measure of the closeness of the relatioinships between its components. A cohesive module

performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.

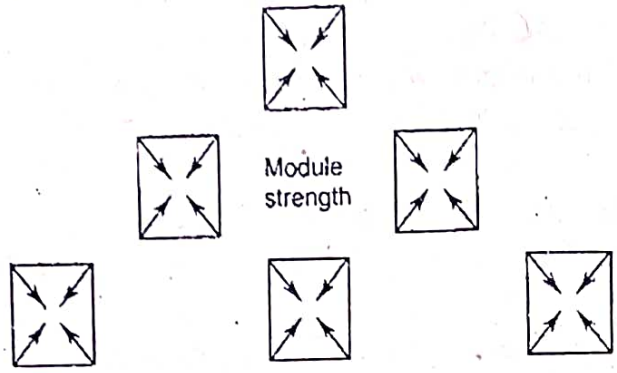


Fig (a) : Cohesion-Strength of Relation within Modules

Coupling : *"Coupling is a measure of interconnection among modules in a software structure."* The coupling between two modules indicates the degree of interdependence between them. If two modules interchange large amount of data, then they are highly interdependent. The degree of coupling between two modules depends on their interface complexity. The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

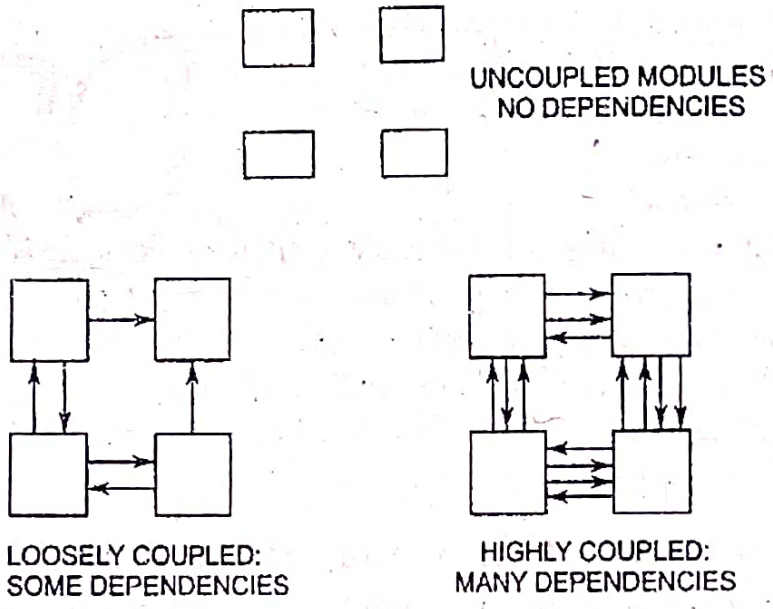


Fig.(b) : Coupling

Types of cohesion : The types of cohesion, in order of the worst to the best type, are as follows :

(i) *Coincidental Cohesion* : Coincidental cohesion is when parts of a module are grouped arbitrarily (at random); the parts have no significant relationship (e.g. a module of frequently used function).

(ii) *Logical Cohesion* : Logical cohesion is when parts of a module are grouped because they logically are categorised to do the samething, even if they are different by nature (e.g. grouping all I/O handling routines).

(iii) *Temporal Cohesion* : Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution

(e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

(iv) *Procedural Cohesion* : Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

(v) *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).

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

Types of Coupling : Different types of Coupling are as follows :

1. *Content Coupling*: Content coupling is when one module modifies or relies on the internal workings of another module. Therefore changing the way the second module produces data will lead to changing the dependent module.

2. *Common Coupling* : Common coupling is when two modules share the same global data. Changing the shared resource implies changing all the modules using it.

3. *External coupling* : It occurs when two modules share an externally imposed data format, communication protocol, or device interface.

4. *Control coupling* : Control coupling is one module controlling the flow of another, by passing it information on what to do.

5. *Data coupling* : Data coupling is when modules share data through.

6. *Message Coupling* : This is the loosest type of coupling. It can be achieved by state decentralization and component communication is done via parameters or message passing.

Unit – III

Q.6. Write the short notes on the following :

(a) Reverse Engineering

(b) Re-Engineering

(c) Forward Engineering

Ans. (a) Reverse engineering : Reverse engineering is the process of analyzing software with the objective of recovering its design and specification.

OR

Reverse engineering is the process followed in order to find difficult ,unknown and hidden information about a software system.

Purpose of reverse engineering : The main purpose of engineering is to recover information from the existing code or any other intermediate documents, an activity that requires program understanding at any level may fall wihin the scope of reverse engineering. The objective of reverse engineering is to derive the design or specification of a system from its source code.

Reverse engineering process : The reverse engineering process is illustrated in fig.(a). The process starts with an analysis phase. During this phase, the system is analyzed using automated tools to discover its structure. In itself, this is not enough to re-create the system design. Engineers then work with the system source code and its structural model. They add

information to this, which they have collected by understanding the system. This information is maintained as a directed graph that is linked to the program source code.

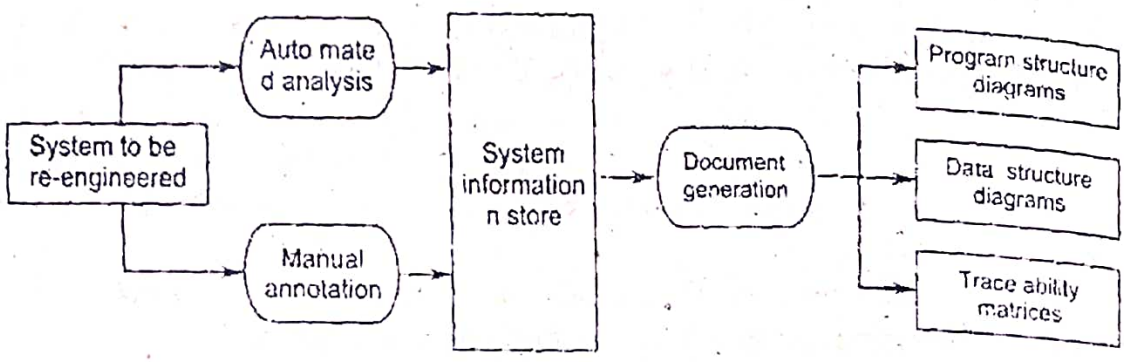


Fig.(a) : The reverse engineering process

Information store browsers are used to compare the graph structure and the code and to annotate the graph with extra information. Documents of various types such as program and data structure diagrams and trace ability metrices can be generated from the directed graph. Trace ability metrices show where entities in the system are defined and referenced. The process of document generation is an iterative one as the design information is used to further refine the information held in the system repository.

Re-engineering : *Re-engineering means to re-implementing systems to make them more maintainable.*

In re-engineering the functionality and system architecture remains the same but it includes re-documenting, organizing and restricting, modifying and updating the system. It is a solution to the problems of system evolution. In other words,

Re-engineering essentially means having a re-look on an entity, such as process, task, design, approach or strategy using engineering principles to bring in radical and dramatic improvement.

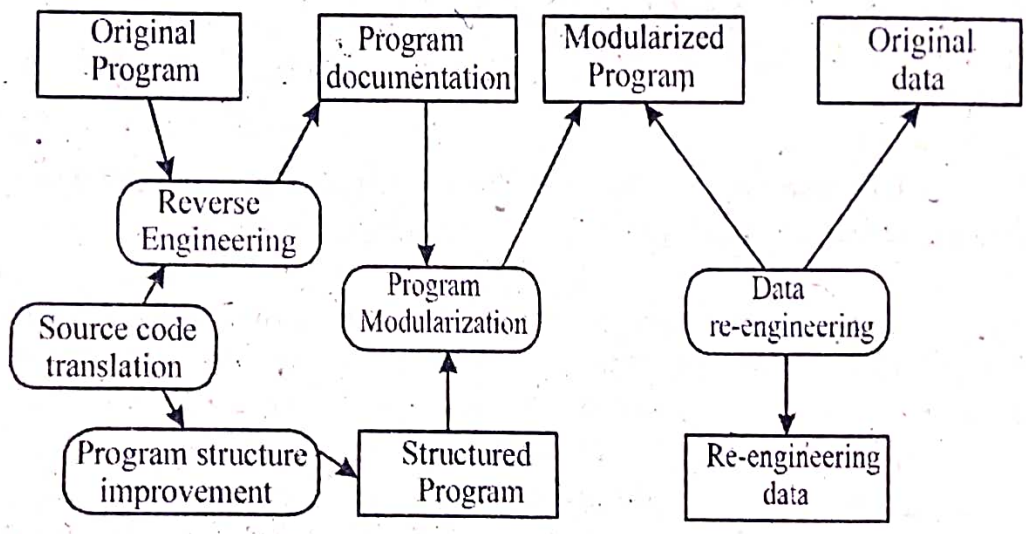


Fig.(b) : Re-engineering Process

The re-engineering approch attacks five parameters, namely : management philosophy, pride, policy, procedures and practises to bring in radical improvement impacting cost, quality, service and speed. When re-engineering principles are applied to business process then it is

called Business Process Re-engineering(BRP). The objective of re-engineering is to produce, new, more maintainable system.

Re-engineering Process : Fig.(b) illustrates a possible re-engineering process. The input to the process is a legacy program and the output is a structured, modularized version of the same program. At the same time as program re-engineering, the data for the system may also be re-engineered.

Re-engineering process includes the following activities :

– *Source code translation :* In it the programming language of an old program is converted into the modern version of the same language or to a new language.

– *Reverse engineering :* In it the program is analyzed and important and useful information are extracted from it which helps to document its functionality.

– *Program structure improvement :* In it control structure of the program is analyzed and modified to make it easier to read and understand.

– *Program modularization :* In it redundancy of any part is removed and related parts are grouped together.

– *Data Re-engineering :* In it the data processed by the program is change to reflect program changes.

Ans.(c) Forward Engineering : The second level of the software re-engineering process is referred to as "forward engineering". The forward engineering procedure corresponds to the customary procedures of the system generation.

The objective of forward engineering is to modify the software part of existing IT systems until they are really state-of-the-art, which (contrary to reverse engineering) may also include functional software modifications.

Typical examples for such modifications are :

- Use of a different programming language.
- Introduction of a new database management system, or
- Transfer of software to a new hardware platform.

When forward engineering is realized in connection with prior reverse engineering, the products generated with reverse engineering remain the basis for the software modification with forward engineering.

Q.7. What is Software Testing ? Give different Software Testing Techniques in detail.

Ans. Refer Q.7(a) of paper May 2017.

Unit – IV

Q.8. Explain the software quality assurance systems in detail.

Ans. Software Quality Assurance : The aim of the software Quality Assurance (SQA) process is to develop high-quality software product. Software Quality Assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained.

Quality assurance is a planned and systematic pattern of all actions necessary to provide adequate confindence that the item or product conforms to established requirements.

The purpose of a software quality assurance group is to provide assurance that the procedures tools, and techniques used during product development and modification are adequate to provide the desired level of confidence in the work products.

The process of software Quality Assurance : (i) Defines the requirements for software controlled system fault/failure detection, isolation, and recovery.

(ii) Reviews the software developement process and products for software error prevention and/or controlled change to reduced functionality states.

(iii) Defines the process for measuring and analyzing defects as well as reliability and main tainability factors.

SQA Objectives : The various objectives of SQA are as follows :

(i) Quality management approach.

(ii) Measurement and reporting mechanisms.

(iii) Effective software engineering technology.

(iv) A procedure to assure compliane with software development standards where applicable.

SQA Goals : The major goals of SQA are as follows :

(i) SQA activities are planned.

(ii) Non-compliance issues that cannot be resolved within the software project are addressed by senior management.

(iii) Adherence of software products and activities to the applicable standards, procedures and requirements is verified objectively.

(iv) Affected groups and individuals are informed of SQA activities and results.

SQA Plan : A plan that defines the quality process and procedures that should be used. This involve selecting and instantiating standards for products and process and defining the required quality attributes of the system.

The SQA Plan provides a road map for instituting software quality assurance. Developed by SQA group (or the software team if a SQA group does not exist), the plan serves as a template for SQA activities that are instituted for each software project.

The quality plan should select those organizational standards that are appropriate to a particular product and developement process. New standards may have to be defined if the project uses new methods and tools.

An outline structure for a quality plan includes :

(i) *Product intoduction* : A description of the product, its intended market and the quality expectations for the product.

(ii) *Product plans* : The critical release dates and responsibilities for the product along with plans for distribution and product servicing.

(iii) *Process descriptions* : The development and service processes that should be used for product development and management.

(iv) *Quality goals* : The quality goals and plans for the product including an identification and justification of crictical product quality attributes.

(v) *Risks and risk management* : The key risks that might affect product quality and the actions to address these risks.

Q.9. Explain Computer Aided Software Engineering in detail.

Ans. Refer Q.9 of paper May 2017.

PRINCIPLES OF SOFTWARE ENGINEERING

May - 2019

Paper Code:-CSE-302-F

Note : ***Attempt five questions in all, selecting one question from each Section. Question No. 1 is compulsory. All questions carry equal marks.***

Q.1.(a) Explain spiral model in detail. **(10)**

Ans. SDLC Spiral Model : The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification : This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

Design : Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct or Build : Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Evaluation and Risk Analysis : Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Fig.(1) is a diagrammatic representation of spiral model listing the activities in each phase.

Limitations of Spiral Model :

(i) No strict standards for software development.

(ii) No particular beginning or end of particular phase.

Advantages of Spiral Model :

(i) It is risk driven model.

(ii) It is very flexible.

(iii) Less documentation is needed.

(iv) It uses prototyping.

(v) It is more realistic model for software development.

Disadvantages of Spiral Model : The spiral model has some disadvantages that need to be resolved before it can be a universally applied life cycle model. The disadvantages include lack of explicit process guidance in determining objectives, constraints, alternatives, relying on risk assessment expertise and providing more flexibility than required for many applications.

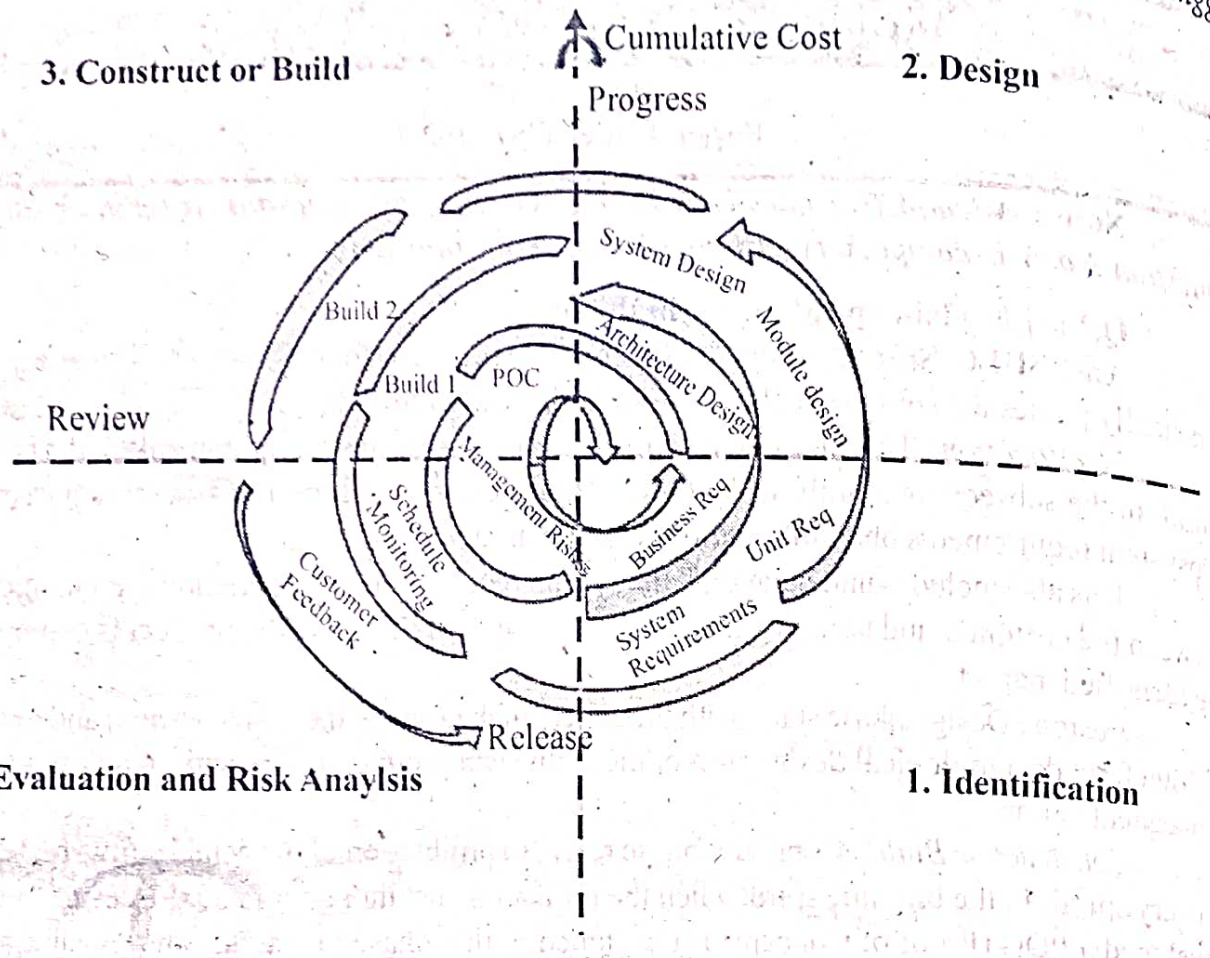


Fig. : (1)

Q.1.(b) Discuss the parameter for the selection of a life cycle model. **(10)**

Ans. The software development life cycle is used to facilitate the development of a large software product in a systematic, well-defined and cost effective way.

A software life cycle model is either a *descriptive or prescriptive* characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. *Descriptive models* may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models. A *Prescriptive model* prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frame works to organize and structure how software development activities should be performed, and in what order.

These two characterizations suggest that there are a variety of purposes for articulating software lifecycle models. These characterizations serve as a :

(i) Guideline to organize, plan, staff, budget, schedule and manage software project work over organizational time, space, and computing environments.

(ii) Prescriptive outline for what documents to produce for delivery to client.

(iii) Basis for determining what software engineering tools and methodologies will be most appropriate to support different lifecycle activities.

(iv) Framework for analyzing or estimating patterns of resource allocation and consumption during the software life cycle.

(v) Basis for conducting empirical studies to determine what affects software productivity, cost, and overall quality.

Section – A

Q.2.(a) Explain the term software crisis. Discuss the causes associated with it. How it lead to the development of software engineering as a discipline? (10)

Ans. Software Crisis : The *software crisis* is characterized by *an inability to develop software on time, on budget and within requirements*. As a result, the delivered software systems are :

(i) Completely unsatisfactory (not as per the specification)
(ii) Extremely late.
(iii) Far over the budget.
(iv) Poorly suited for intended user of the system.
(v) Projects running over-time.
(vi) Software was of low qualtity.
(vii) Software often did not meet requirements.
(viii) Projects were unmanageable & code difficult to maintain.

Various **causes associated with crisis** are as follows :

(i) Lack of communication between software developers and users.
(ii) Increase in cost of software compared to hardware.
(ii) Increase in the size of software.
(ii) Increased complexity of the porblem area.
(ii) Project management problem.
(ii) Lack of understanding of the problem and its environment and
(ii) High optimistic estimates regarding software development time and cost
(ii) Duplication of efforts due to absent of automation in most of the software development activites.

Development of software engineering as a discipline : Software engineering principles have evolved over the past more than fifty years from art to an engineering discipline. It can be shown with the help of the following fig.

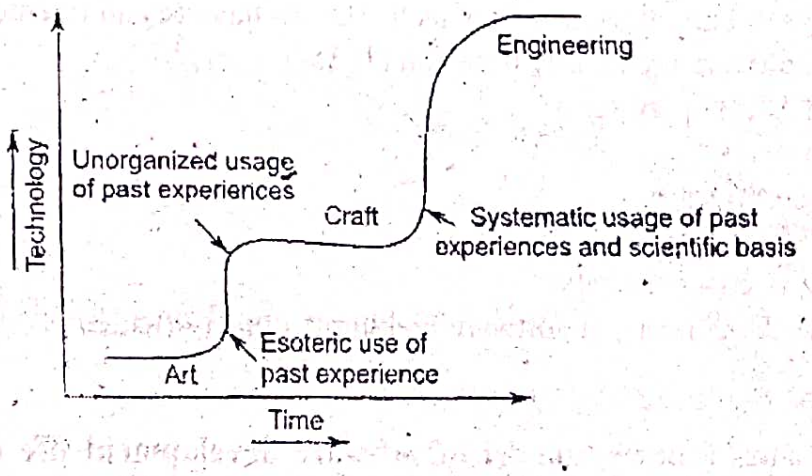


Fig. : Evolution of art to an engineering discipline

Development in the field of software and hardware computing make a significant change in the twentieth century. We can decide the software development process into four areas :

(i) Early Era : During the early eras general-purpose hardware became common place.

Software, on the other hand, was custom-designed for each application and had a relatively limited distribution. Most software was developed and ultimately used by the same person or organization.

In the era the software are mainly based on (1950-1960)

- Limited distribution
- Custom software
- Batch Orientation

(ii) Second Era : The second era to computer system evolution introduced new concepts of human machine interaction. Interactive techniques opened a new world of application and new levels of hardware and software sophistication. Real time software deals with the changing environment and one other is multi-user in which many users can perform or work on a software at a time.

In this era the software are mainly based on (1960-1972)

- Multi-user
- Data base
- Real time
- Product Software
- Multiprogramming

(iii) Third Era : In the earlier age the software was custom designed and limited distribution but in this era the software was consumer designed and the distribution is also not limited. The cost of the hardware is also very low in this era.

In this era the software are mainly based on (1973-1985)

- Embedded intelligence
- Consumer Impact
- Distributed Systems
- Low cost hardware

(iv) Fourth Era : The fourth era of computer system evolution moves us away from individual computers and computer programs and toward the collective impact of computers and software. As the fourth era progresses, new technologies have begun to emerge.

In this the software are mainly based on (1985-)

- Powerful Desktop systems
- Expert systems
- Artificial intelligence
- Parallel Computing
- Object oriented technology

At this time the concept of software making is object oriented technology or network computing etc.

Q.2.(b) Discuss iterative model of software development life cycle. (10)

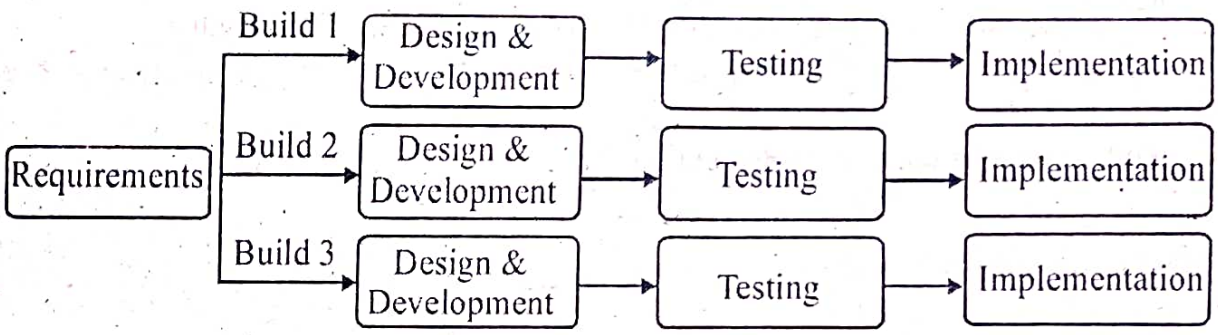
Ans. SDLC Iterative Model : Following is the pictorial representation of Iterative and Incremental model:

This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested

enhancements may evolve with time.

- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.



Advantages of Iterative Enhancement Model : The various advantages of following the approach of the iterative enhancement are as follows :

(i) The feedbacks from early increments improve the later stages.

(ii) The possibility of changes in requirement is reduced because of the shorter time span between the design of a component and its delivery.

(iii) Users get benefits earlier than with a conventional approach.

(iv) Early delivery of some useful components improves cash flow, because you get some return on investment early on.

(v) Smaller sub-projects are easier to control and manage.

(vi) 'Gold-plating', that is the requesting of features that are unnecessary and not in fact used, is less as users will know that they get more than one bite of the cherry if a feature is not in the current increment then it can be included in the next.

(vii) The project can be temporarily abandoned if more urgent work crops up.

(viii) Job satisfaction is increased for developers who see their labours bearing fruit at regular, short, intervals.

Disadvantages of Iterative Enhancement Model : The various disadvantages of iterative enhancement model have been put forward :

(i) Software breakage, that is, later increments may require modifications to earlier increments.

(ii) Programmers may be more productive working on one large system than on a series of smaller ones.

(iii) Some problems are difficult to divide into functional units (modules), which can be incrementally developed and delivered.

(iv) Testing of modules result into overhead and increased cost.

Q.3.(a) Explain COCOMO Model for estimating cost of software. (10)

Ans. COCOMO model : COCOMO stands for *Constructive Cost Model*. It was introduced by Barry Boehm in 1981. It is perhaps the best known and most throughly documented of all software cost estimation models. It provides the following three levels of models :

(i) *Basic COCOMO model :* The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions :

$$\text{Effort} = a_1 \times (KLOC)^{a_2} \quad PM$$

$$T_{dev} = b_1 \times (Effort)^{b_2} \quad Months$$

Where, *KLOC* is the estimated size of the software product expressed in kilo lines, Code a_1, a_2, b_1, b_2 are constants of software products.

T_{dev} is the estimated time to develop the software, expressed in months.

Effort is the total effort required to develop the software product, expressed in person-month (PM).

Person months curve is shown in fig.(1).

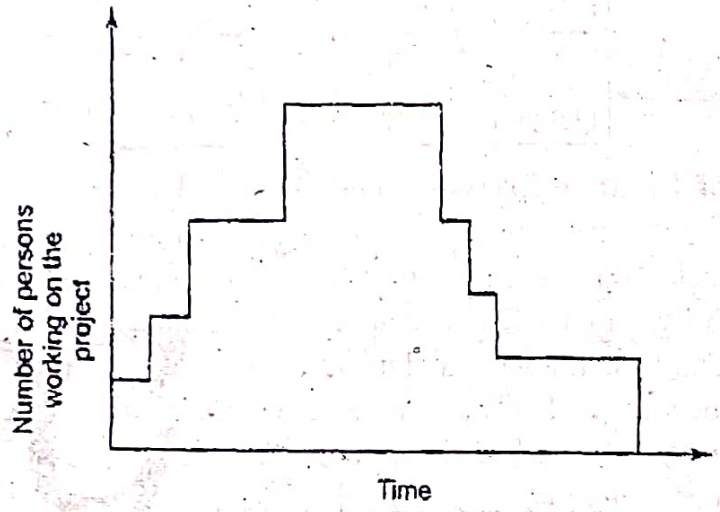


Fig.(1) : Person-month curve

Estimation of Development effort :

Organic: Effort = $2.4(\text{KLOC})^{1.05}$ PM

Semidetached : Effort = $3.0(\text{KLOC})^{1.12}$ PM

Embedded : Effort = $3.6(\text{KLOC})^{1.20}$ PM

Fig.2(a) shows a plot of estimated effort versus size for various product sizes.

From fig.2(a), we can observe that the effort is almost linearly proportional to the size of the software product.

Estimation of Development time :

Organic: $T_{dev} = 2.5(\text{Effort})^{0.38}$ Months

Semidetached: $T_{dev} = 2.5(\text{Effort})^{0.35}$ Months

Embedded: $T_{dev} = 2.5(\text{Effort})^{0.32}$ Months

Fig.2(b) shows the plot is development time versus product size.

From fig.2(b), we can observe that the development time is sub linear function of the size of the product.

(ii) *Intermediate COCOMO Model :* The intermediate COCOMO model recognizes this fact and refines the intial estimate obtained through the BASIC COCOMO expression by using a set of 15 cost drivers (multipliers) based on various attributes of software development.

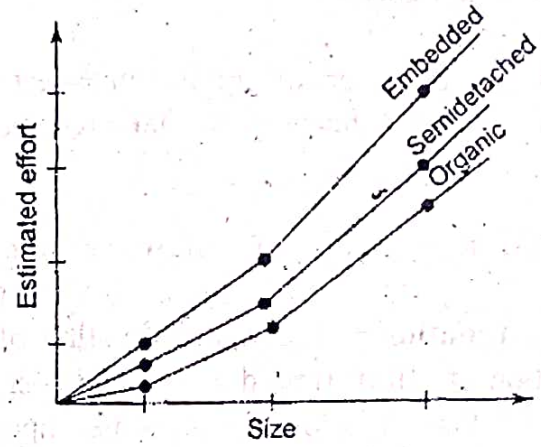


Fig.2(a) : Effort versus size

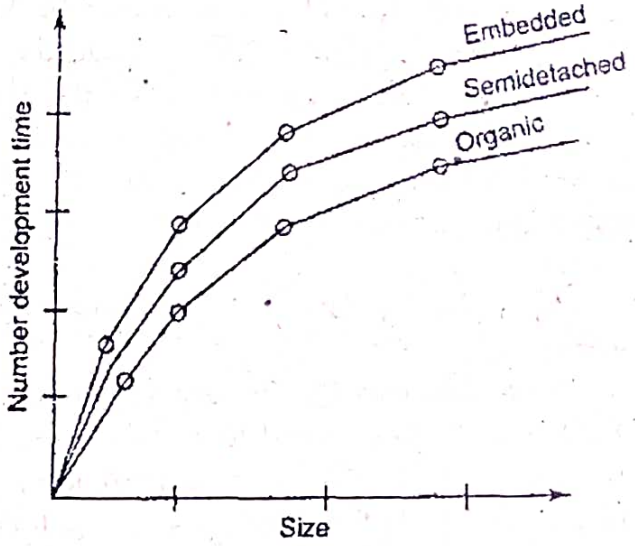


Fig.2(b) : Development Time versus size

Table : COCOMO Intermediate Cost drivers

Driver type	Code	Cost Driver
Product attributes	RELY	Required software reliability
	DATA	Database size
	CPLX	Product complexity
Computer attributes	TIME	Execution time constraints
	STOR	Main storage constraints
	VIRT	Virtual machine volatility-degree to which the operating system changes
	TURN	Computer turn around time
Personnel attributes	ACAP	Analyst capability
	AEXP	Application experience
	PCAP	Programmer capability
	VEXP	Virtual machine (i.e.,operation system) experience
	LEXP	Programming language experience
Project attributes	MODP	Use of modern programming practices
	TOOL	Use of software tools
	SCED	Required development schedule.

(iii) *Complete COCOMO Model :* The short-comings of both basic and intermediate COCOMO models are that they :

– Consider a software product as a single homogeneous entity.

– However, most large systems are made up of several smaller sub-systems. Some sub-systems may be considered as organic type, some embedded, etc. For some the reliability requirements may be high and so on.

– Cost of each sub-system is estimated separately.
– Costs of the sub-systems are added to obtain total-cost.
– Reduces the margin or error in the final estimate.

A Large amount of work has been done by Boehm to capture all significant aspects of a software development. It offers a means for processing all the project characteristics to construct a software estimate.

Q.3.(b) Discuss project size estimation metrics of software project management. **(10)**

Ans. Metrics for Software project size estimation : Accurate estimation of the problem size is fundamental to satisfactory estimation of effort, time duration and cost of a software project. In order to be able to accurately estimate the project size, some important metrics should be defined in terms of which the project size can be expressed. The size of a problem is obviously not the number of bytes that the source code occupies. It is neither the byte size of the executable code. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product. Currently two metrics are popularly being used widely to estimate size; Lines of Code (LOC) and Function Point (FP). The usage of each of these metrics in project size estimation has its own advantages and disadvantages.

Lines of Code (LOC) : LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.

Determining the LOC count at the end of a project is a very simple job. However, accurate estimation of the LOC count at the beginning of a project is very difficult. In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted. To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.

Function Point (FP) : Function point metric was proposed by Albrecht [1983]. This metric overcomes many of the shortcomings of the LOC metric. Since its inception in late 1970s, function point metric has been slowly gaining popularity. One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification. This is in contrast to the LOC metric, where the size can be accurately determined only after the product has fully been developed.

The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. Each function when invoked reads some input data and transforms it to the corresponding output data. For example, the issue book feature (as shown in Fig.) of a Library Automation Software takes the name of the book as input and displays its location and the number of copies available. Thus, a computation of the number of input and the output data values to a system gives some indication of the number of functions supported by the system.

Albrecht postulated that in addition to the number of basic functions that a software performs, the size is also dependent of the number of files and the number of interfaces.

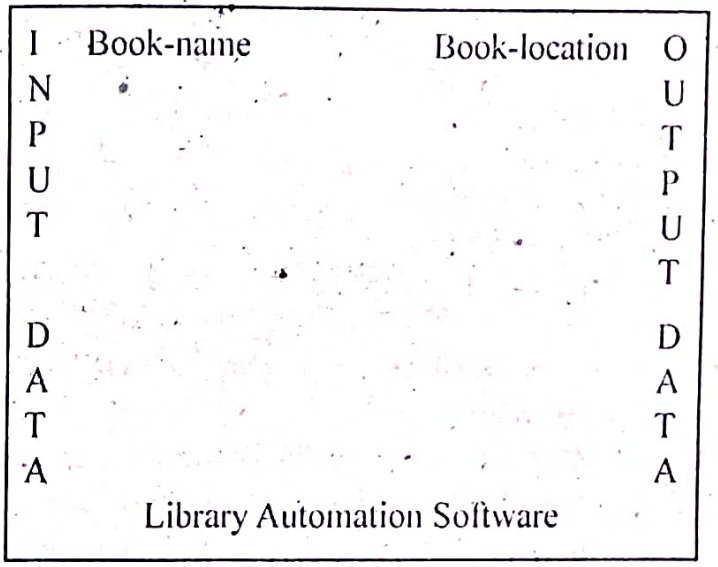


Fig. System function as a map of input data to output data

Besides using the number of input and output data values, function point metric computes the size of a software product (in units of functions points or FPs) using three other characteristics of the product as shown in the following expression. The size of a product in Function Points (FP) can be expressed as the weighted sum of these five problem characteristics. The weights associated with the five characteristics were proposed empirically and validated by the observations over many projects. Function point is computed in two steps. The first step is to compute the Unadjusted Function Point (UFP).

UFP = (Number of inputs)*4 + (Number of outputs)*5 +
(Number of inquiries)*4 + (Number of files)*10 + (Number of interfaces)*10

Number of Inputs : Each data item input by the user is counted. Data inputs should be distinguished from user inquiries. Inquiries are user commands such as print-account-balance. Inquiries are counted separately. It must be noted that individual data terms input by the user are not considered in the calculation of the number of inputs, but a group of related inputs are considered as a single input. For example, while entering the data concerning employee to employee pay roll software; the data items name, age, address, phone number, etc., are together considered as a single input. All these data items can be considered to be related, since they pertain to a single employee.

Number of Output : The outputs considered refer to reports printed, screen outputs, error messages produced, etc. While outputting the number of outputs the individual data items within a report are not considered, but a set of related data items is counted as one input.

Number of Inquiries : Number of inquiries is the number of distinct interactive queries which can be made by the users. These inquiries are the user commands which require action by the system.

Number of Files : Each logical file is counted. A logical file means groups of logically related data. Thus, logical files can be data structures or physical files.

Number of Interfaces : Here the interfaces considered are the interfaces used to exchange information with other systems. Examples of such interfaces are data files on tapes, disks, communication links with other systems etc.

Once the Unadjusted Function Point (UFP) is computed, the Technical Complexity Factor (TCP) is computed next. TCF refines the UFP measure by considering fourteen other factors such as high transaction rates, throughput, and response time requirements, etc. Each of these 14 factors is assigned from 0 (not present or no influence) to 6 (strong influence). The resulting numbers are summed, yielding the total degree of influence (DI). Now, TCF is computed as (0.65+0.01*DI). As DI can vary from 0 to 70, TCF can vary from 0.65 to 1.35. Finally, FP = UFP*TCF.

Section – B

Q.4. Describe the basic components of a Data Flow Diagram (DFD). Also create DFD to describe Library Information System. **(20)**

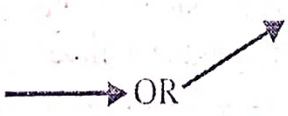
Ans. Data Flow Diagram is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output.

There are different types of symbols used to construct DFDs. The meaning of each symbol is explained below:

(1) Function symbol : A function is represented using a circle. This symbol is called a process or a bubble or performs some processing of input data.

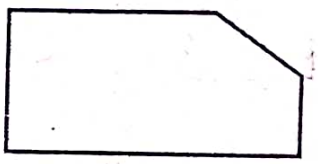
(2) External entity : A square defines a source of destination of system data. External entities represent any entity that supplies or receives information from the system but is not a part of the system.

(3) Data flow symbol : A directed arc or arrow is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.



(4) Data store symbol : A data store symbol is represented using two parallel lines. A logical file can represent either a data store symbol, which can represent either a data structure, or a physical file on disk. Each data store is connected to a process by means of a data flow symbol. The direction of the data flow arrow shows whether data is being read from or written into a data store.

(5) Output symbol : It is used to represent data acquisition and production during human computer-interaction.



DFD for library information system : In this DFD the whole system is represented with the help of input, processing and output. The input can be :

(i) Student requests for a book hence Book request.

(ii) To show identity of the student he/she has to submit his/her Library card, hence Library card. The processing unit can be globally given as

Library information system : The system will produce following outputs :

(i) The demanded book will be given to student. Hence Book will be the output.

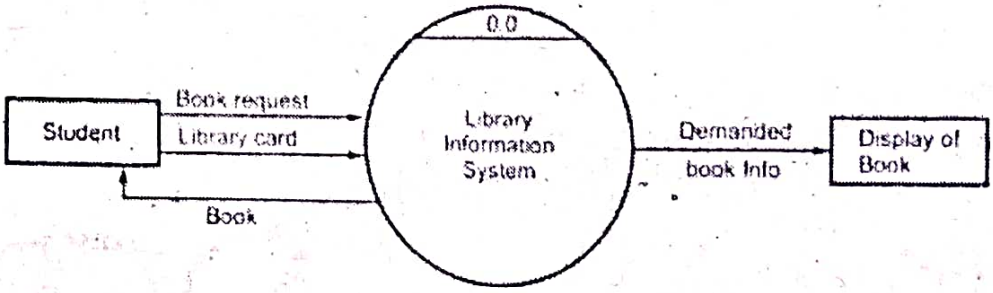


Fig.(1) : Level 0 DFD (Context level DFD)

(ii) The library information system should display demanded book information which can be used by customer while selecting the book.

Level 1 DFD :

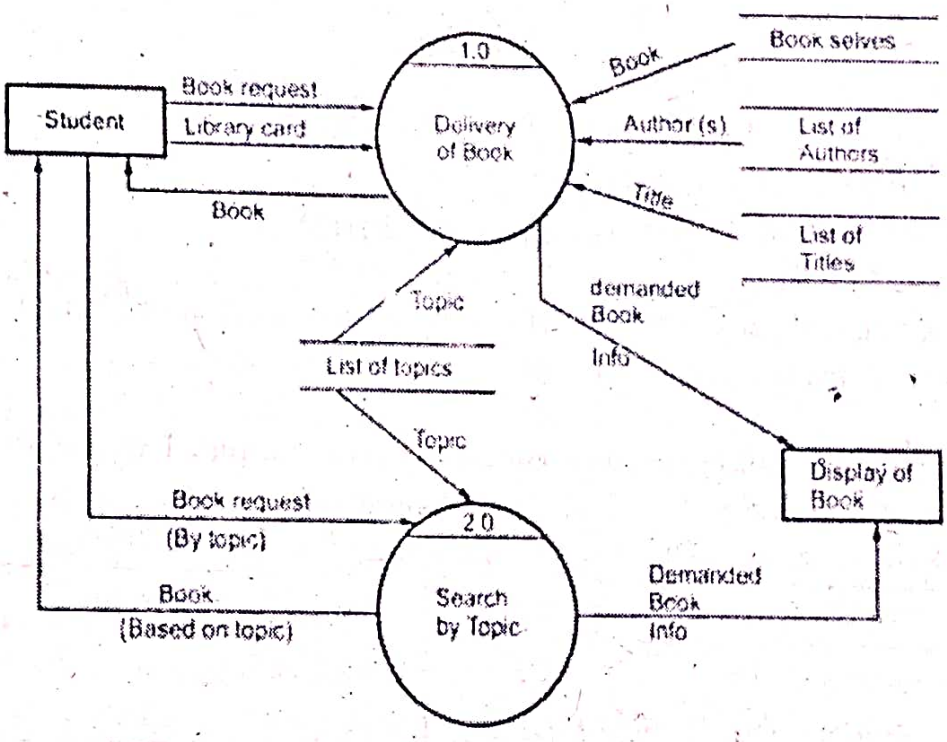


Fig.(2) : Level 1 DFD

In this level, the system is exposed with more processing details. The processes that need to carries out are :

(i) Delivery of Book.

(ii) Search by Topic.

These processes require certain information such as List of Authors, List of Titles, List of Topics, the book shelves from which books can be located. This kind of information is represented by data store.

Level 2 DFD :

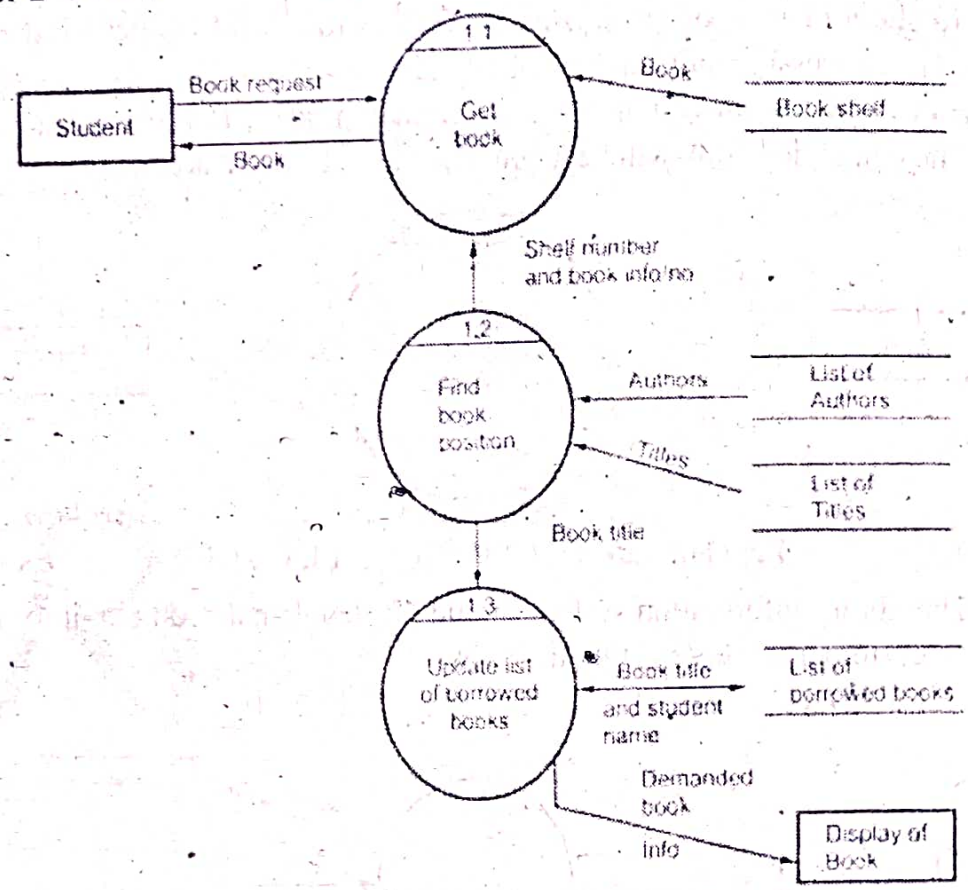


Fig.(3) : Level 2 DFD

Out of Scope : The purchasing of new books/replacement of old books or charging a fine all these maintenance activities are not considered in this system.

Q.5.(a) Differentiate between cohesion and coupling. Explain various types of cohesion. Which one is best and which one is worse? (10)

Ans. Difference between Cohesion and Coupling in Tabular Form :

Cohesion	Coupling
Cohesion is the indication of the relationship within the module.	Coupling is the indication of the relationship between modules.
Cohesion shows the module's relative functional strength.	Coupling shows relative independence among the modules.

Cohesion is a degree (quality) to which a component/module focuses on a single thing.	Coupling is a degree to which a component/module is connected to the other modules.
While designing you should strive for high cohesion i.e. a cohesive component/module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.	While designing you should strive for low coupling i.e. dependency between modules should be less.
Cohesion is the kind of natural extension of data hiding, for example, the class having all members visible with a package having default visibility.	Making private fields, private methods and nonpublic classes provide loose coupling.
Cohesion is Intra-Module Concept.	Coupling is the Inter-Module Concept.

Cohesion : *"Cohesion is a natural extension of information hiding concept."* Cohesion is a measure of the relative functional strength of a module. The cohesion of a component is a measure of the closeness of the relatioinships between its components. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.

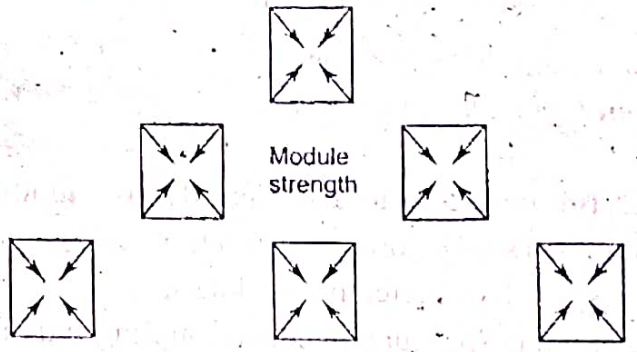


Fig (a) : Cohesion-Strength of Relation within Modules

Types of cohesion : The types of cohesion, in order of the worst to the best type, are as follows :

(i) Coincidental Cohesion : Coincidental cohesion is when parts of a module are grouped arbitrarily (at random); the parts have no significant relationship (e.g. a module of frequently used function).

(ii) Logical Cohesion : Logical cohesion is when parts of a module are grouped because they logically are categorised to do the samething, even if they are different by nature (e.g. grouping all I/O handling routines).

(iii) Temporal Cohesion : Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

(iv) Procedural Cohesion : Procedural cohesion is when parts of a module are grouped

because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

(v) 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).

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

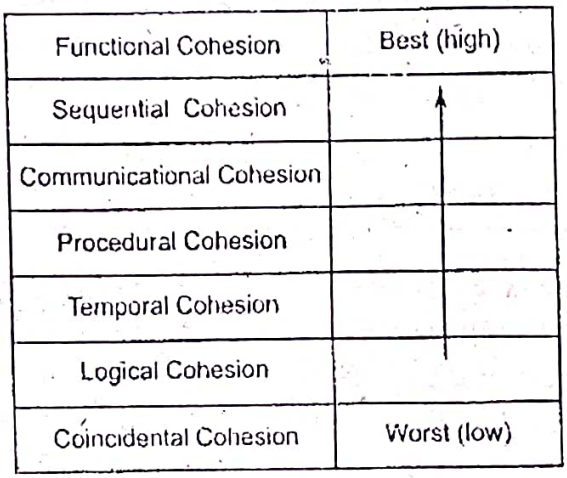


Fig (b) : The Types of Module Cohesion

Best Cohesion: Functional Cohesion
Worst Cohesion: Coincidental Cohesion

Q.5.(b) Discuss the design heuristic for effective modularity. **(10)**

Ans. Design Heuristics : The program structure can be manipulated according to the following design heuristics to achieve effective modularity :

(1) Evaluate the program structure to reduce coupling and improve cohesion. Once, the program structure has been developed, modules may be exploded or imploded towards improving module independence.

(2) Attempt to minimize structures with high Fan-Out; Strive for Fan-In as Depth Increases.

(3) Keep scope of effect of a module within the scope of control of that module. The scope of effect of module A is defined as all other modules that are affected by a decision made in module A. The scope of control of module A is all modules that are sub-ordinate to module A.

(4) Evaluate Module Interfaces to reduce complexity and redundancy and improve consistency.

(5) Define modules whose function is predictable, but avoid modules that are overly restrictive.

(6) Strive for single-entry-single-exit modules, avoiding Pathological connections. (Pathological Connection refers to branches or references into the middle of a module).

(7) Package software based on design constraints and portability requirements.

Section – C

Q.6.(a) Differentiate between databases and data warehouse. (10)

Ans. Some differences between a database and a data warehouse:

- A database is used for Online Transactional Processing (OLTP) but can be used for other purposes such as Data Warehousing.
- A data warehouse is used for Online Analytical Processing (OLAP). This reads the historical data for the Users for business decisions.
- In a database the tables and joins are complex since they are normalized for RDMS. This reduces redundant data and saves storage space.
- In data warehouse, the tables and joins are simple since they are de-normalized. This is done to reduce the response time for analytical queries.
- Relational modeling techniques are used for RDMS database design, whereas modeling techniques are used for the Data Warehouse design.
- A database is optimized for write operation, while a data warehouse is optimized for read operations.
- In a database, the performance is low for analysis queries, while in a data warehouse, there is high performance for analytical queries.
- A data warehouse is a step ahead of a database. It includes a database in its structure.

Q.6.(b) Explain different architectural design issues. (10)

Ans. Architectural Design : Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design include :

- Gross decomposition of the system into major components;
- Allocation of functional responsibilities to components;
- Component interfaces;
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth;
- Communication and interaction between components.

Aspects of a system's architecture may be specified in its requirements, but this is less often the case than with interface design.

Architectural design adds important details ignored during interface design. Design of the internals of the major components is ignored until the last phase of design.

Q.7.(a) What are different levels of testing? (10)

Ans. Levels of Testing : There are three levels of testing *i.e.*, individual module to the entire software are system.

Unit Testing : In unit testing individual components are tested to ensure that they operate correctly. It focuses on verification effort. On the smallest unit of software design, each component is tested independently without other system componets.

because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

(v) 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).

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

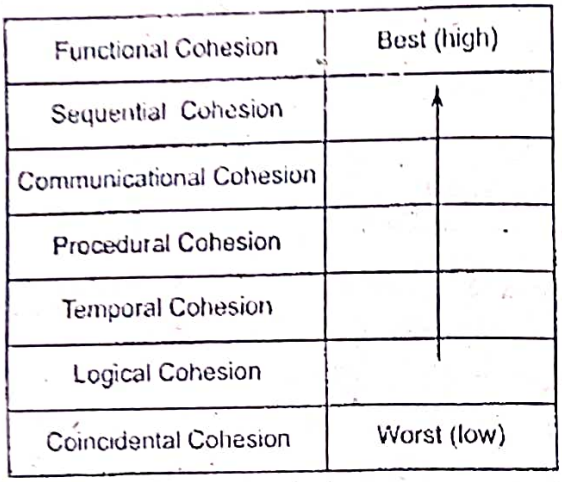


Fig (b) : The Types of Module Cohesion

Best Cohesion: Functional Cohesion
Worst Cohesion: Coincidental Cohesion

Q.5.(b) Discuss the design heuristic for effective modularity. (10)

Ans. Design Heuristics : The program structure can be manipulated according to the following design heuristics to achieve effective modularity :

(1) Evaluate the program structure to reduce coupling and improve cohesion. Once, the program structure has been developed, modules may be exploded or imploded towards improving module independence.

(2) Attempt to minimize structures with high Fan-Out; Strive for Fan-In as Depth Increases.

(3) Keep scope of effect of a module within the scope of control of that module. The scope of effect of module A is defined as all other modules that are affected by a decision made in module A. The scope of control of module A is all modules that are sub-ordinate to module A.

(4) Evaluate Module Interfaces to reduce complexity and redundancy and improve consistency.

(5) Define modules whose function is predictable, but avoid modules that are overly restrictive.

(6) Strive for single-entry-single-exit modules, avoiding Pathological connections. (Pathological Connection refers to branches or references into the middle of a module).

(7) Package software based on design constraints and portability requirements.

Section – C

Q.6.(a) Differentiate between databases and data warehouse. **(10)**

Ans. Some differences between a database and a data warehouse:

- A database is used for Online Transactional Processing (OLTP) but can be used for other purposes such as Data Warehousing.
- A data warehouse is used for Online Analytical Processing (OLAP). This reads the historical data for the Users for business decisions.
- In a database the tables and joins are complex since they are normalized for RDMS. This reduces redundant data and saves storage space.
- In data warehouse, the tables and joins are simple since they are de-normalized. This is done to reduce the response time for analytical queries.
- Relational modeling techniques are used for RDMS database design, whereas modeling techniques are used for the Data Warehouse design.
- A database is optimized for write operation, while a data warehouse is optimized for read operations.
- In a database, the performance is low for analysis queries, while in a data warehouse, there is high performance for analytical queries.
- A data warehouse is a step ahead of a database. It includes a database in its structure.

Q.6.(b) Explain different architectural design issues. **(10)**

Ans. Architectural Design : Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design include :

- Gross decomposition of the system into major components;
- Allocation of functional responsibilities to components;
- Component interfaces;
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth;
- Communication and interaction between components.

Aspects of a system's architecture may be specified in its requirements, but this is less often the case than with interface design.

Architectural design adds important details ignored during interface design. Design of the internals of the major components is ignored until the last phase of design.

Q.7.(a) What are different levels of testing? **(10)**

Ans. Levels of Testing : There are three levels of testing *i.e.*, individual module to the entire software are system.

Unit Testing : In unit testing individual components are tested to ensure that they operate correctly. It focuses on verification effort. On the smallest unit of software design, each component is tested independently without other system componets.

There are number of reasons in support of unit testing than testing the entire product.

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

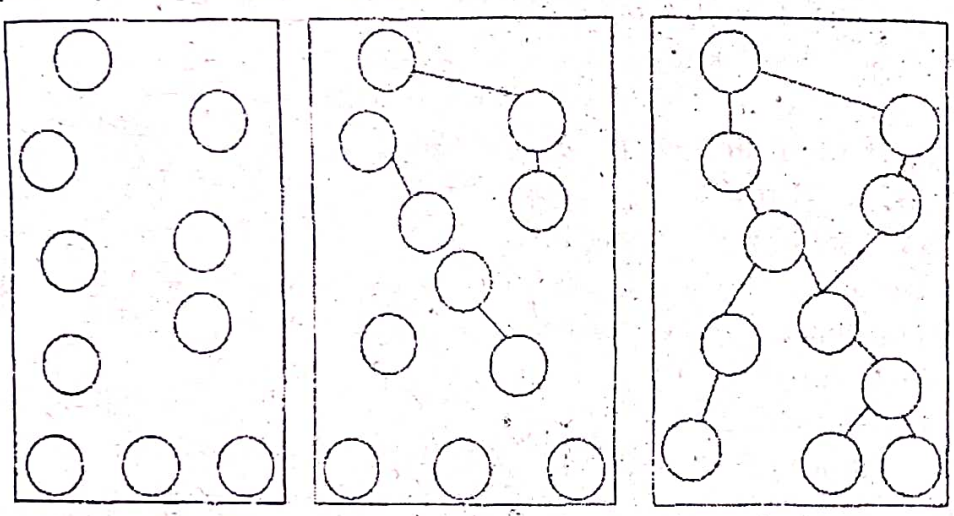
– The module is small enough that we can attempt to test it in some demonstrably exhaustive fashion.

– Confusing interactions of mulitple errors in widely different parts of the software are eliminated.

Integration Testing : The second level of testing is called integration testing. Integration testing is a systematic technique for contructing the program structure while at the same time conducting tests to uncover errors is associated with interfacing.

In this testing many units tested modules are combined into sub-systems, which are then tested. The goal here is to see if the modules can be integrated properly.

Objective of IntegrationTesting : The primary objective of integration testing is to test the module interfaces in order to ensure that 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.



Approaches to Integration Testing : The various approaches, which are used for the integration testing, are :

1. Incremental Appoach
2. Top-down Integration
3. Bottom-up Integration
4. Regression Testing
5. Smoke Testing
6. Sandwich Integration

System Testing : The sub-systems are integrated to make up the entire system. The testing process is concerned with finding errors that result from unanticipated interactions between sub-systems and system components. It is also concerned with vailating that the system needs its functional and non-functional requirements.

There are essentially three main kinds of system testing.

1. Alpha testing
2. Beta testing
3. Acceptance testing

1. *Alpha Testing :* Alpha testing refers to the system testing carried out by the test team within the development organization.

2. *Beta Testing :* Beta testing is the system testing performed by a selected group of friendly customers.

3. *Acceptance Testing :* Acceptance testing is the system performed by the customer to determine whether to accept or reject the delivery of the system.

Q.7.(b) What is software testing? Also discuss software testing strategies.(10)

Ans. Software testing : Software testing is the process of executing a program with the intention of finding errors in the code. It is the process of exercising or evaluating a system or system components by manual or atuomatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

The objects of testing is to show incorrectness and testing is considered to succeed when an error is detected. An error is a conceptual mistake made by either the programmer or the designer or a discrepancy between a computed value and a theoretically correct value. A fault is a specific manifestation of an errror. An error may be the cause of several faults. A failure is the inability of a system or component to perform its required function within the specified limits. A failure may be produced when a fault is executed or exercised.

Testing should not be a distinct phase in system development but should be applicable throughout the design, development and maintenance phases.

Software Testing Strategies : Software testing strategy provides a road map for the software developer, quality assurance organization, and the customer.

A strategy must provide guidance for the practitioner and a set of milestones for the manager. Progress must be measurable and problems must surface as soon as possible.

In order to conduct a proper and through set of tests, the types of testing mentioned below should be performed in the order in which they are described.

(1) Unit Testing
(2) Integration Testing
(3) Functional Testing
(4) Regression Testing
(5) Systems and Acceptance Testing.

However, some system or hardware can happen concurrently, with software testing.

Unit Testing **:** Unit testing procedure utilizes the white-box method and concentrates on testing individual programming units. These units are somtimes referred to as modules or atomic modules and they represent the smallest programming entity.

Unit testing is essentially a set of path test performed to examine the many different paths through the modules. These types of tests are conducted to prove that all paths in the program are solid and without errors and will not cause abnormal termination of the program or other undesirable results.

Integration Testing **:** Integration testing focuses on testing multiple modules working together. Two basic types of integration are usually used :

(i) Top-down Integration

(ii) Bottom-up Integration

Above types of integration are discussed below :

Top down Integration : As the term suggest, starts at the top of the program hierarchy and travels down its branches. This can be done in either depth-first (shortest path down to the deepest level) or breadth-first (across the hierarchy, before proceeding to the next level).

The main advantage of this type of integration is that the basic skeleton of the program/ system can be seen and tested early.

The main disadvantage is the use of program stubs until the actual are written. This basically limits the up-flow of information and therefore does not provide for a good test of the top-level modules.

Bottom-up Integration : This type of integration has the lowest level modules built and tested before its utilized by its calling module.

This method has a great advantage in unconvering errors in critical modules early.

Main disadvantage is the fact that most or many modules must be build before a working program can be presented.

Functional Testing **:** Functional testing verifies that an application does what it is supposed to do and doesnot do what it shouldn't do.

For example, if you are functionally testing a word processing application, a partial list of checks you would perform minimally includes creating, saving, editing, spell checking and printing documents.

Function testing usually includes testing of all the interfaces and should therefore involve the clients in the process. Because every aspect of the software system is being and how will conduct the tests and what exacutly will be tested.

Regression Testing : Regression testing is the process of running a subset of previously executed integration and function tests to ensure that program changes have not degraded the system. The regressive phase concerns the effects of newly introduced changes on all the previously integrated code. Problems arise when errors made in incorporating new functions affects the previously tested functions, which are common in large systems.

Regression testing may be conducted manually or using automated tools. The basic regression resting approach is to incorporate selected test cases into a regression bucket that is run periodically to find regression problems. In many organizations regression testing consists of running all the functional tests every few months. This generally delays the regression problem detection and results in significant rework after every regression run.

It is wise to accumalte a comprehensive regression bucket and also define a subset of test cases. The full bucket in run occasionally, but the subset is run against every spin. The spin subset consist of all test cases for recently integrated functions and a selected sample from the full regression bucket. Depending on the success history of test subsets and the complexity of the program the test cases are selected from the subset.

Systems and Acceptance Testing **:** Final stage of the testing process should be system Testing and Acceptance Testing. It is concerned with the execution of test cases to evaluate the whole system with respect to the user's requirement. A system test checks for unexpected interaction between the units and modules, and also evaluates the system for compliance with functional requirements.

An acceptance test is the process of executing the test cases agreed with the customer as being an adequate representation of user requirments. These are often called Black Box or Functional tests.

Section – D

Q.8.(a) Explain the features of ISO 9001 requirements. (10)

Ans. ISO 9001 standard : ISO 9001 is a Quality assurance standard that is specific to software engineering. It specifies 20 standards with which an organization must comply for an effective implementaion of the quality assurance system. It is an international quality management system. ISO 9001 is mainly related to the software industry. It lay down the standards for designing, developing, servicing the producing of standard quality of goods. It is also applicable to most software development organizations.

The salient features of ISO 9001 requirements are as follows :

(i) All documents concerned with the development of a software product should be

properly managed, authorized and controlled.

(ii) Proper plans should be prepared and then progress against these plans should be monitored.

(iii) Important documents should be independently checked and reviewed for effectiveness and correctness.

(iv) The product should be tested against its specification.

Q.8.(b) Discuss software quality assurance. Also explain software quality assurance activity. **(10)**

Ans. Software Quality Assurance : The aim of the software Quality Assurance (SQA) process is to develop high-quality software product. Software Quality Assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained.

Quality assurance is a planned and systematic pattern of all actions necessary to provide adequate confindence that the item or product conforms to established requirements.

The purpose of a software quality assurance group is to provide assurance that the procedures tools, and techniques used during product development and modification are adequate to provide the desired level of confidence in the work products.

The process of software Quality Assurance : (i) Defines the requirements for software controlled system fault/failure detection, isolation, and recovery.

(ii) Reviews the software developement process and products for software error prevention and/or controlled change to reduced functionality states.

(iii) Defines the process for measuring and analyzing defects as well as reliability and main tainability factors.

SQA Objectives : The various objectives of SQA are as follows :

(i) Quality management approach.

(ii) Measurement and reporting mechanisms.

(iii) Effective software engineering technology.

(iv) A procedure to assure compliance with software development standards where applicable.

SQA Goals : The major goals of SQA are as follows :

(i) SQA activities are planned.

(ii) Non-compliance issues that cannot be resolved within the software project are addressed by senior management.

(iii) Adherence of software products and activities to the applicable standards, procedures and requirements is verified objectively.

(iv) Affected groups and individuals are informed of SQA activities and results.

Software quality assurance activity : SQA is the process of evaluating the quality of a product and enforcing adherence to software product standards and procedures. It is an umbrella activity that ensures conformance to standards and procedures throughout the SDLC of a software product. These include:

(i) Formulating a quality management plan : The quality management plan identifies the quality aspects of the software product to be developed. It helps in planning checkpoints for work products and the development process. It also tracks changes made to the development process based on the results of the checks.

(ii) Applying software engineering techniques : It helps the software designer to achieve high quality specification. The designer gathers information using techniques such as interviews. Using the information gathered, the designer prepares project estimation with the help of techniques such as SLOC, FP estimation.

(iii) Conducting formal technical reviews : Formal technical review is conducted to assess the quality and design of the prototype. It is a meeting with the technical staff to discuss the quality requirements of a software product and its design quality. It helps in detecting errors at an early phase of development.

(iv) Applying a multi-tiered testing strategy : Software testing is a critical task of SQA activity, which aims at error detection. Unit testing is the first level of testing. The subsequent levels of testing are integration and system level testing. There are various testing strategies followed by an organization.

(v) Enforcing process adherence : It emphasizes the need for process adherence during product development. The development process should also adhere to procedures defined for product development. It is a combination of product evaluation and process monitoring.

(vi) Controlling change : It combines human procedures and automated tools to provide a mechanism for change control. It ensures software quality by formalizing requests for change, evaluating the nature of change and controlling the impact of change.

(vii) Measuring impact of change : Changes need to be measured and monitored. Changes are measured using software quality metrics. Software quality metrics helps in estimating the cost and resource requirements. It is essential to measure quality and then compares it with the established standards.

(viii) Performing SQA audits : They scrutinize the software development process by comparing it with the established processes. It ensures that the proper control is maintained over the documents required during SDLC.

(ix) Keeping records and reporting : Keeping records and reporting ensure the collection and circulation of information relevant to SQA. The result of reviews, audits, testing etc. are reported and compiled for future reference.

Q.9. Explain computer Aided Software engineering in detail. **(20)**

Ans. CASE stands for computer aided software engineering.

CASE is a tool which aids a software engineer to maintain and develop software. The workshop for software engineering is called in Integrated Project Support Environment (IPSE) and the tool set that fills the workshop is called CASE.

CASE is a computer aided software engineering technology CASE is an automated support tool for the software engineers in any software engineering process.

Software engineering mainly includes the following processes :

1. Translation of user needs into software requirements.
2. Transaction of software requirements into design specification
3. Implementation of design into code
4. Testing of the code
5. Documentation

CASE technology provides software process support by automating some process activities and by providing information about the software, which is being developed. Examples of activities, which can be automated using CASE, include:

1. The development of graphical system models as part of the requirements specification or the software design.
2. Understanding a design using a data dictionary, which holds information about the entities and relations in a design.
3. The generation of user interfaces from a graphical interface description, which is created interactively by the user.
4. Program debugging through the provision of information about an executing program.
5. The automated translation of programs from an old version of a programming language such as COBOL to a more recent version.

The use of Computer Aided Software Engneering (CASE) tool reduce the effort of development of achieving quality goals and managing change and configuration throughout the product life cycle. It also help the project manager, the software developer and other key personnel to improve their productivity in the development team.

CASE Tools : CASE Tools are software programs that are designed to assist human programmers with the complexity of the processes and the artifacts of software engineering.

They constitute the laws of and the automated tools that aid in the synthesis, analysis, modelling, or documentation of software.

The schematic diagram of CASE tools is shown in fig.

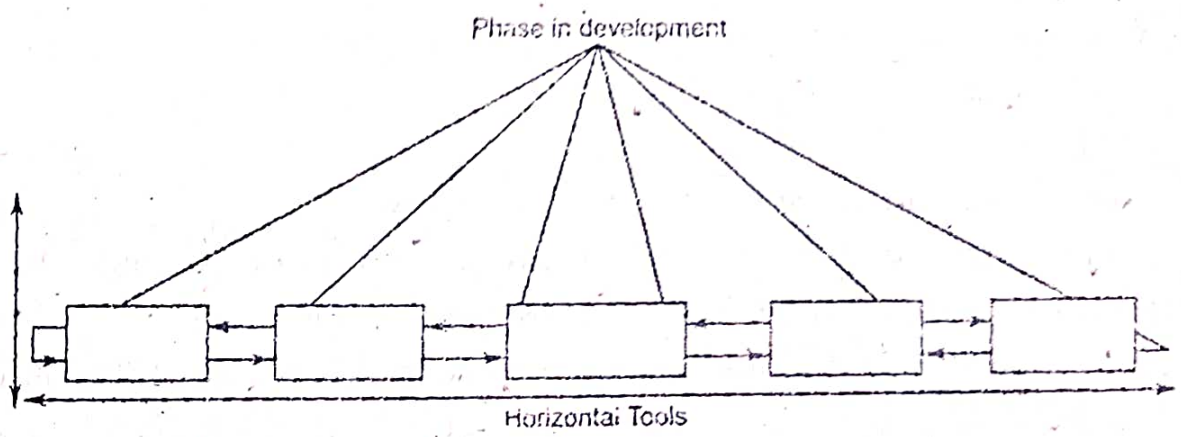


Fig. : Categories of case tools

Smith and Oman have defined CASE tools which are divided into the following two categories :

(i) Vertical CASE tools

(ii) Horizontal CASE tools

(i) Vertical CASE Tools : Vertical CASE tools provide support for certain activities within a single phase of the software life cycle.

There are two subcategories of vertical CASE tools :

– *First category :* It is the set of tools that are within one phase of life cycle. These tools are important so that development in each phase can be as quick as possible.

– *Second category :* It is a tool that is used in more than one phase, but does not support moving from one phase to the next. These tools ensure that the developer does move on the next phase as appropriate.

(ii) Horizontal CASE tools : These tools support automated transfer of information between the phases of a life cycle. These tools include project management, configuration management tools and integration services.

Various **advantages of CASE tools** are as follows :

(i) Improved productivity

(ii) Better documentation

(iii) Improved accuracy

(iv) Intangible benefits

(v) Improved quality

(vi) Reduced lifetime maintenance

(vii) Opportunity to non-programmers

(viii) Reduced cost of software

(ix) Produce high quality and consistent documents

(x) Impact on the style of a working of company

(xi) Reduce the drudgery in a software engineer's work

(xi) Increase speed of processing

(xii) Easy to program software

Various **disdvantages of CASE tools** are as follows :

(i) *Purchasing of case tools is not and easy task :* Its cost is very high. Due to this reason small software development firm do not invest in case tools.

(ii) *Learning curve :* In general cases programmer productivity may fall in initial phase of implementation as user need time to learn this technology.

(iii) *Tool mix :* It is important to make proper selection of case tools to get maximum benefit from the case tool, so wrong selection may lead to wrong result.