- Every change requires that the complete system be re-verified. If we do not have the ability to run a mechanized regression test, this is going to be an extremely tedious and expensive process.

Myth 3: Computers provides greater reliability than the devices they replace :

- It is true that software does not fail in the traditional sense. There are no limits to how many times a given piece of code can be executed before it 'wears out'.
- In any event, the simple expression of this myth is that our general ledgers are still not perfectly accurate, even though they have been compu-terized.
- Back in the days of manual accounting systems, human errors was a fact of life. Now, we have software error as well.

Myth 4: Increasing software reliability will increases safety :

- It is possible for software to be perfectly reliable and still do the wrong thing. One of the things we emphasize in our process maturity consulting is the importance of identifying defects as early as possible in the product life cycle.
- Finding and repairing a defect in a requirement is approximately ten times more economical than finding the defect during coding.
- Beyond that, defects that escape the requirements process can become part of the software itself.

Myth 5: Testing software or 'proving' software correct can remove all the errors :

- Software testing can only be as good as the requirements specification it is based on. Since the requirements specification is written in 'natural language', there are all sorts of opportunities for interpretation.
- Also, modern software has so many theoretical execution paths that it is impossible at a practical level to test them all.
- The best we can hope for is a statistically meaningful coverage level of the most anticipated paths.

Myth 6 : Reusing software increases safety :

- Certainly reusing software can increase reliability. But the trouble goes back to the requirements issues under myth #5.
- Despite everyone's best efforts, the completely 'generic' modules that are written almost certainly have some degree of context dependency. Put those same modules in a different context and unpredictable things can happen.
- This myth is particularly troubling because of the false sens security that code re-use can create.
- Code reuse is a very powerful tool that can yield dram improvement in development efficiency, but it still requires anal to determine its suitability and testing to determine if it worked.

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY

Q.10. Explain layered approach in software engineering.

CT : W-06(6M), S-11(2M), CS : W-10(7M), S-11(6M

OR Comment on "Software engineering - a layered technology"

CT : S-08(5M), S-10,14, W-12(4M), W-11(7M), S-12(2M

CS : W-12(4M), S-13(6M

OR Explain the layered view of software engineering in detail.

CS : S-12, W-13(6M

OR How the quality will be focused with the help of process, meth and tools? Justify. *CT : S-13(7M*

Ans. Software Engineering - A Layered Technology :

- Software engineering is the application of a systematic, disciplin quantifiable approach to the development, operation and maintenan of software.
- Referring to figure, any engineering approach must rest on organizational commitment to quality.

Fig. Software Engineering Layer

- The software engineering is layered in following categories :

(1) A quality focus :

Total quality management leads to the development of increasingl more mature approaches to software engineering. The bedrock th supports software engineering is a quality focus.

(2) Process :

- The foundation for software engineering is the process laye Software engineering process is the glue that holds the technolo layers together and enables rational and timely development computer software.

- Process defines a framework for a set of key process areas (KPAs) that must be established for effective delivery of software engineering technology.
- The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured and change is properly managed.

(3) **Methods :**

- Software engineering methods provide the technical how-to's for building software.
- Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing and support.
- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modelling activities and other descriptive techniques.

(4) **Tools :**

- Software engineering tools provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development.
- Hardware and a software engineering database (a repository containing important information about analysis, design, program construction and testing) to create a software engineering environment analogous to computer-aided design/engineering) for hardware.

SOFTWARE PROCESS FRAMEWORK

Q.11. Explain software process.

Ans. Software process :

- When we build a product or system, it's important to go through a series of predictable steps-a road map that helps us to creates a timely, high-quality result. The road map that we follow is called a software process.
- Software engineers and their managers adapt the process to their needs and then follow it. In addition, the people who have requested the software play a role in the software process.
- It is important because it provides stability, control and organization to an activity that can if left uncontrolled become quite chaotic.
- At a detailed level, the process that we adopt depends on the software we are building.
- One process might be appropriate for creating software for an aircraft avionics system, while an entirely different process would be indicated for the creation of a Web site.
- From the point of view of a software engineer, the work products are the programs, documents and data produced as a consequence of the software engineering activities defined by the process.
- A number of software process assessment mechanisms enable organizations to determine the 'maturity' of a software process.
- However, the quality, timeliness and long-term viability of the product we build are the best indicators of the efficacy of the process that we use.
- A software process defines the approach that is taken as software is engineered.
- But software engineering also encompasses technologies that populate the process technical methods and automated tools.
- More important, software engineering is performed by creative, knowledgeable people who should work within a defined and mature software process that is appropriate for the products they build and the demands of their market place.

Q.12. Explain the common process framework with the help of diagram. ***CT : W-11(4M), S-12(7M)***

OR Explain common process framework in detail. ***CS : S-09(6M)***

OR Explain software process framework with diagram. Also explain umbrella activities. ***CS : S-14(9M)***

Ans. Software process framework :

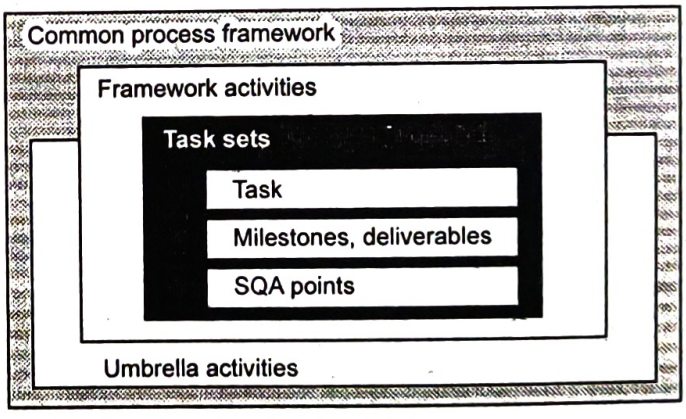


Fig. Software Process

- A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- Software projects are executed for building a variety of systems in different types of organizations. Each project requires a suitable process definition. Therefore, there is a need for a common process framework (CPF) that defines the broad activities applicable across all projects in the organization.
- A CPF acts as a starting point for defining a process suitable for the project.
- The CPF is a generic process framework and the activities defined in the CPF capture the best practices suited to an organization.
- In order to use the CPF, it is important to establish adaptation guidelines that are used to select the task sets suitable for each project.
- A number of task sets each a collection of software engineering work tasks, project milestones, work products and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
- Umbrella activities such as software quality assurance, software configuration management and measurement overlay the process model.
- Umbrella activities are independent of any one framework activity and occur throughout the process.
- The following five generic framework activities can be used during the development of small programs, the creation of large web applications and for the engineering of large complex computer based systems.

(1) Modeling : Modeling activity encompasses the creation of models that allow the developer and the customer to better understand software requirements, specification and the design that will achieve those requirements.

(2) Construction : Construction activity combines code generation and the testing i.e. required to uncover errors in the code.

(3) Communication : Communication framework activity involves heavy communication and collaboration with the customer, encompasses requirements gathering and other related activities.

(4) Deployment : The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

(5) Planning : Planning activity establishes a plan for the software engineering work that follows. Planning describes the technical tasks to be conducted, the resources that will be required, the risks that are likely the work products to be produced.

Q.13. Explain briefly the various capability levels given by CMMI for assessing each process area. **CS/IS-10(5M)**

Ans. Capability levels :

- In recent years, there has been a significant emphasis on "process maturity".
- The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity.
- To determine an organization current state of process maturity, the SEI uses an assessment that results in a five point grading scheme.
- The grading scheme determines compliance with a capability maturity model (CMM) that defines key activities required at different levels of process maturity.
- The SEI approach provides a measures of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner.

(1) Level 1: Initial :

The software process is charact-erized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

(2) Level 2: Repeatable :

Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

(3) Level 3: Defined :

- The software process for both management and engineering activities is documented, standardized and integrated into an organization wide software process.
- All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

ARE PROCESS MODEL

Q.16. What are the different process model?

OR Write a short note on software process model.

Ans. Software process model :

- To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process methods, and tools layers and the generic phases. This strategy is often referred to as a process model or a software engineering paradigm.
- A process model for software engineering is chosen based on the nature of the project and applications the methods and tools to be used and the controls and deliverables that are required.
- All software development can be characterized as a problem solving loop Fig. (a) in which four distinct stages are encountered as Status quo, problem definition, technical development and solution integration.

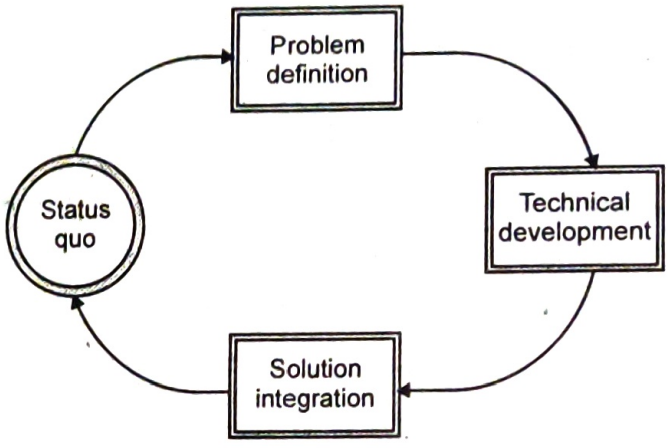


Fig.(a) The phases of a problems solving loop

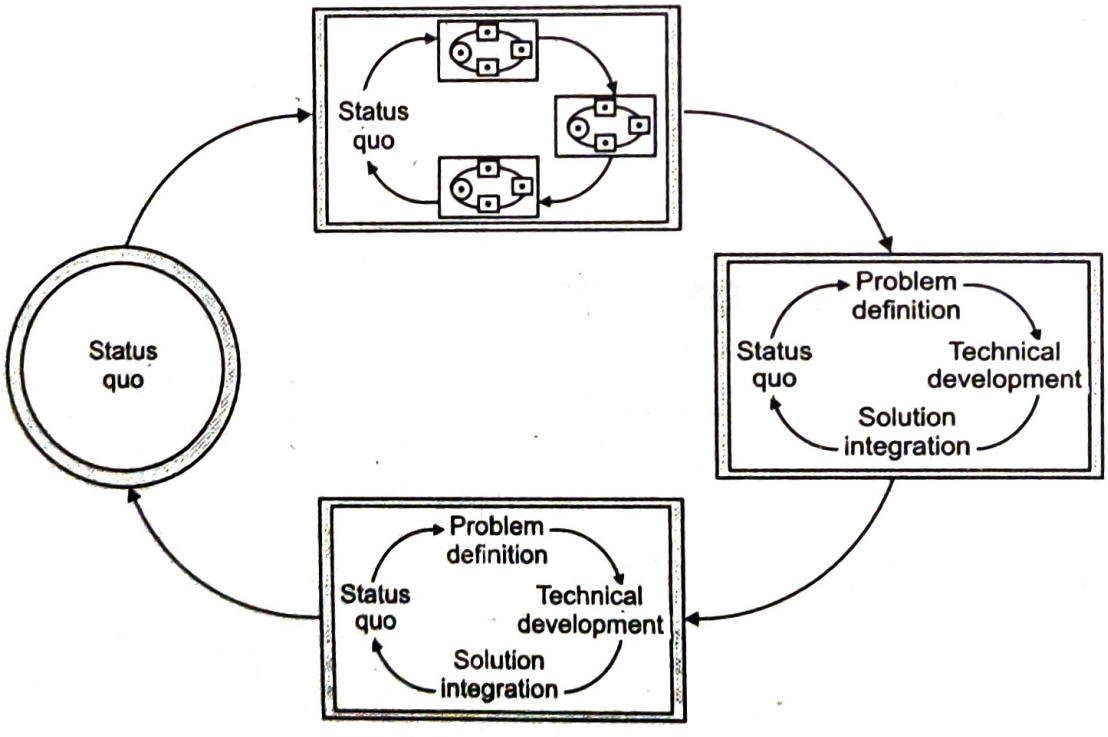


Fig.(b) The phases within phases of a problems solving loop

- Status quo represents the current state of affairs, problem definition identifies the specific problem to be solved, technical development solves the problem through the applications of some technology and solution integration delivers the results (e.g. document, programs, data, new business function, new product) to those who requested the solution in the first place.
- This problem solving loop applies to software engineering work at many different levels of resolution.
- It can be used at the macro level when the entire application is considered, at a mid-level when program components are being engineered and even at the line of code level.
- Therefore, a fractal representation can be used to provide an idealized view of process.
- In Fig.(b) each stage in the problem solving loop contains an identical problem solving loop, which contains still another problem solving loop (this continues to some rational boundary, for software a line of code).
- The different process model are as follows :

(1) The Linear Sequential Model.

(2) The Prototyping Model.

(3) The RAD Model.

(4) Evolutionary Software Process Models.

(5) The Incremental Model.

(6) The Spiral Model.

(7) The WINWIN Spiral Model.

(8) The Concurrent Development Model.

THE WATERFALL MODEL

Q.17. Write a note on waterfall model for software development. What are problems encountered with it? *CT : S-06(6M), W-08,09(7M)*

OR Explain waterfall model of software development in detail.

CS : S-09(4M), W-09(7M), W-10(6M)

OR What is software lifecycle model? *CT : W-10(4M)*

Ans. Waterfall model :

- Sometimes it is also called the classic life cycle or the linear sequential model.
- It suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support.
- The principal stages of the waterfall model directly reflect the fundamental development activities :
- The waterfall model is consistent with other engineering process models and documentation is produced at each phase.

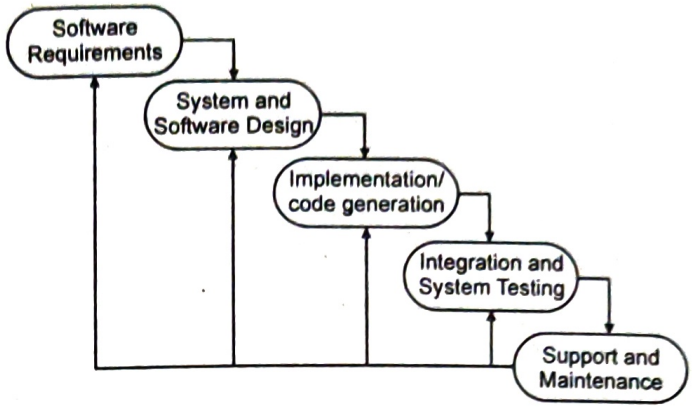


Fig: The Waterfall model

(1) Software requirements :

- The requirements gathering process is intensified and focused specifically on software.
- To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance and interface.
- Requirements for both the system and the software are documented and reviewed with the customer.

(2) Software design :

- Software design is actually a multi-step process that focuses on four distinct attributes of a program : data structure, software architecture, interface representations and procedural (algorithmic) detail.
- The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.

(3) Code generation / implementation :

- The design must be translated into a machine-readable form.
- The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

Testing :

- Once code has been generated, program testing begins.
- The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals.
- Testing is conducting to uncover errors and ensure that defined input will produce actual results that agree with required results.

(5) Support and maintenance :

- Software will undoubtedly undergo change after it is delivered to the customer.
- Change will occur because errors have been encountered, because the software must be adapted to accommodate changes in its external environment (e.g. a change required because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements.
- Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

Drawbacks of Waterfall model:

(i) Waterfall model does not allow iteration of phases. Hence it is not reflecting the problem solving nature of software development.

(ii) The whole project is integrated only at the end. Hence integration is tough job.

(iii) Customer can preview the system only at the end. No prototype are developed and shown to the user.

Q.18. What are the advantages and disadvantages of Waterfall model?

CT : W-10(4M), CS : S-09(4M)

Ans. Advantages :

(1) Relatively simple to understand.

(2) Each phase of development proceeds sequentially.

(3) Allows managerial control where a schedule with deadlines is set for each stage of development.

(4) Helps in controlling schedules, budgets and documentation.

Disadvantages :

(1) Requirements need to be specified before the development proceeds.

(2) The changes of requirements in later phases of the waterfall model cannot be done. This implies that once the software enters the testing phase, it becomes difficult to incorporate changes at such a late phase.

(3) No user involvement and working version of the software is available when the software is being developed.

(4) Does not involve risk management.

(5) Assumes that requirements are stable and are frozen across the project span.

INCREMENTAL PROCESS MODELS

Q.19. Explain with suitable diagram the Incremental model, an evolutionary s/w process model.

Ans. Incremental model :

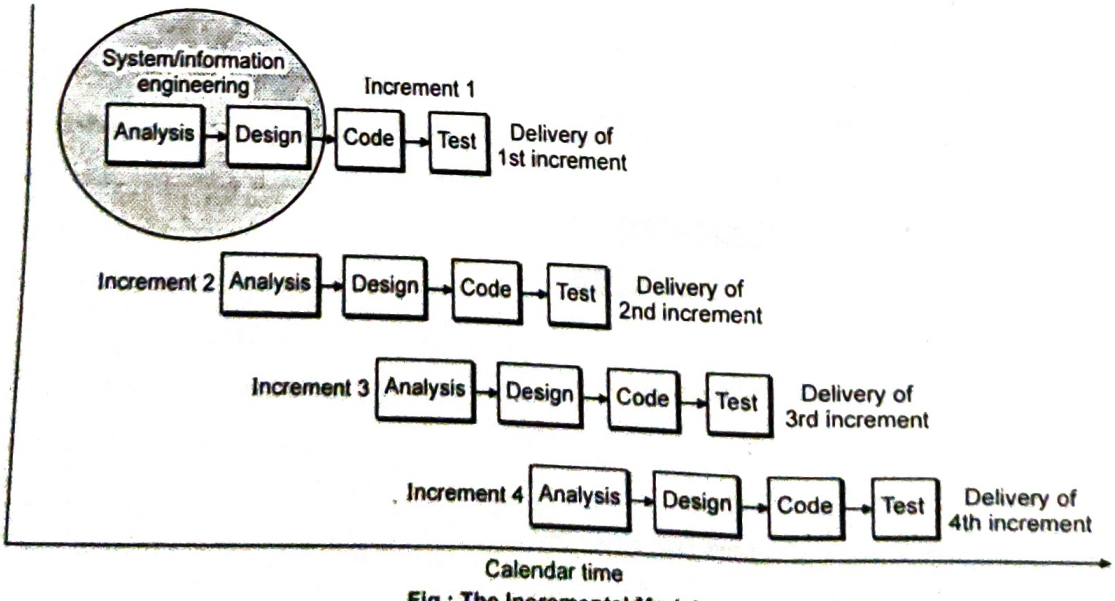


Fig : The Incremental Model

- The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.
- The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software.
- When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features remain undelivered and the core product is used by the customer. 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
- The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature.
- Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.
- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project.
- In addition, increments can be planned to manage technical risks. For example, a major system might require the availability of new hardware that is under development and whose delivery date is uncertain.

Q.20. State the advantages and disadvantages of Incremental model.

Ans. Advantages :

(1) Requirements are relatively stable and may be better understood with each increment.

(2) It is more responsive to user needs than the waterfall model.

(3) The project can be stopped any time after the first cycle and leave a working product.

(4) Project management may be easier for smaller, incremental projects.

Disadvantages :

(1) The majority of requirements must be known in the beginning.

(2) Cost and schedule overruns may result in an unfinished system.

(3) Operations are impacted as each new release is deployed.

(4) Users are required to learn how to use a new system with each deployment.

Q.21. Explain RAD model of software process.

CT : W-06,07(6M), S-11, W-12(3M), CS : S-11(5M)

OR Write a short note on RAD model. ***CT : S-07(4M)***

Ans. Rapid Application Development (RAD) :

- Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.
- The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
- If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g. 60 to 90 days).
- The RAD approach encompasses the following phases :

(1) **Business modelling :**

The information flow among business functions is modelled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

(2) **Data modelling :**

- The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business.
- Characteristics (called attributes) of each object are identified and the relationships between these objects defined.

(3) **Process modelling :**

- The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function.
- Processing descriptions are created for adding, modifying, deleting or retrieving a data object.

(4) Application generation :

- RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components or create reusable components .
- In all cases, automated tools are used to facilitate construction of the software.

(5) Testing and turnover :

Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time.

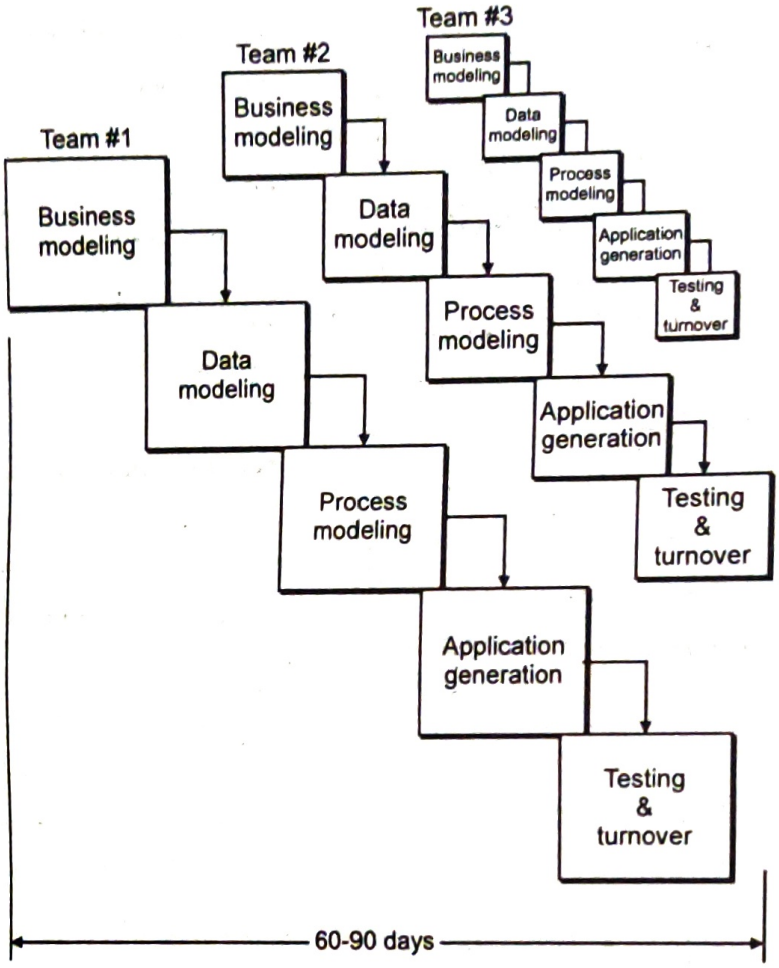


Fig : RAD Model

Q.22. State advantages and disadvantages of RAD model.

CS : S-11(2M), CT : W-06,07(4M)

OR State advantages of RAD model. *CT : S-11(1M)*

OR State disadvantages of RAD model. *CT : W-12(2M)*

OR What are the problems with RAD model? *CS : W-13(4M)*

Ans. Advantages :

(1) The RAD process enables to create a fully functional system within very short time period.

(2) Key objective is for fast development & delivery of a high quality system at a relatively low investment cost.

(3) Active user involvement is imperative.

Disadvantages :

(1) For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

(2) RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.

(3) Not all types of applications are appropriate for RAD. If a system cannot be properly modularized, building the components necessary

OR **Explain framework activities of spiral model adopted for entire life-cycle.** **CT : S-13(6M)**

Ans. **Spiral model :**

- The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
- Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype.
- During later iterations, increasingly more complete versions of the engineered system are produced.
- A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions.
- Figure shows spiral model that contains six task regions as follows :

(1) **Customer communication :** Tasks required to establish effective communication between developer and customer.

(2) **Planning :** Tasks required to define resources, timelines and other project related information.

(3) **Risk analysis :** Tasks required to assess both technical and management risks.

(4) **Engineering :** Tasks required to build one or more representations of the application.

(5) **Construction and release :** Tasks required to construct, test, install and provide user support.

(6) **Customer evaluation :** Tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

- Each of the regions is populated by a set of work tasks, called a task set, that are adapted to the characteristics of the project to be undertaken.
- For small projects, the number of work tasks and their formality is low.
- For larger, more critical projects, each task region contains more work tasks that are defined to achieve a higher level of formality.

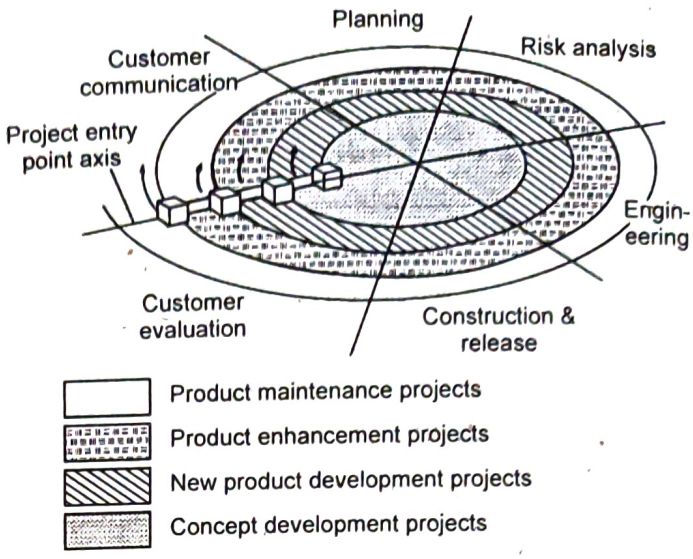


Fig : A typical spiral model

- In all cases, the umbrella activities (e.g. software configuration management and software quality assurance) are applied.
- As this evolutionary process begins, the software engineering term moves around the spiral in a clockwise direction, beginning at the center.
- The first circuit around the spiral might result in the development of a product specification ; sub-sequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan.
- Cost and schedule are adjusted based on feedback derived from customer evaluation.
- In addition, the project manager adjusts the planned number of iterations required to complete the software.
- Unlike classical process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer software.
- An alternative view of the spiral model can be considered by examining the project entry point axis, also shown in Figure.
- Each cube placed along the axis can be used to represent the starting point for different types of projects.
- A concept development project starts at the core of the spiral and will continue (multiple iterations occur along the spiral path that bounds the central shaded region) until concept development is complete.

- If the concept is to be developed into an actual product, the process proceeds through the next cube (new product development project entry point) and a new development project is initiated.
- The new product will evolve through a number of iterations around the spiral, following the path that bounded the region that has somewhat lighter shading than the core.
- In essence, the spiral, when characterized in this way, remains operative until the software is retired.
- There are times when the process is dormant, but whenever a change is initiated, the process starts at the appropriate entry point (e.g., product enhancement).

Applications :

(i) The spiral model of software engineering is typically used by big software companies for large projects. These projects are developed in phased manner and each spiral adds in more minute details of development at each stage.

(ii) For a typical shrink-wrap application, the spiral model might mean that we have a rough-cut of user elements (without the polished/pretty graphics) as an operable application, add features in phases, and at some point, add the final graphics. This is the main reason why spiral model is used most often in large software projects. Hence spiral model is considered as meta model.

Q.28. Give advantages and disadvantages of spiral model.

CT : W-07, S-09 (4M), S-12(2M), CS : W-12(2M), S-13(2M)

Ans. Advantages :

(1) High amount of risk analysis hence, avoidance of Risk is enhanced.

(2) Good for large and mission-critical projects.

(3) Strong approval and documentation control.

(4) Additional functionality can be added at a later date.

(5) Software is produced early in the software life cycle.

Disadvantages :

(1) Can be a costly model to use.

(2) Risk analysis requires highly specific expertise.

(3) Project's success is highly dependent on the risk analysis phase.

(4) Doesn't work well for smaller projects.

Q.29. Explain the WINWIN Spiral Model for software development process in detail

CT : W-13(4M)

Ans. WINWIN spiral model :

- The objective of this activity is to elicit project requirements from the customer. In an ideal context, the developer simply asks the customer what is required and the customer provides sufficient detail to proceed.
- The best negotiations strive for a "win-win" result. That is, the customer wins by getting the system or product that satisfies the majority of the customer's needs and the developer wins by working to realistic and achievable budgets and deadlines.

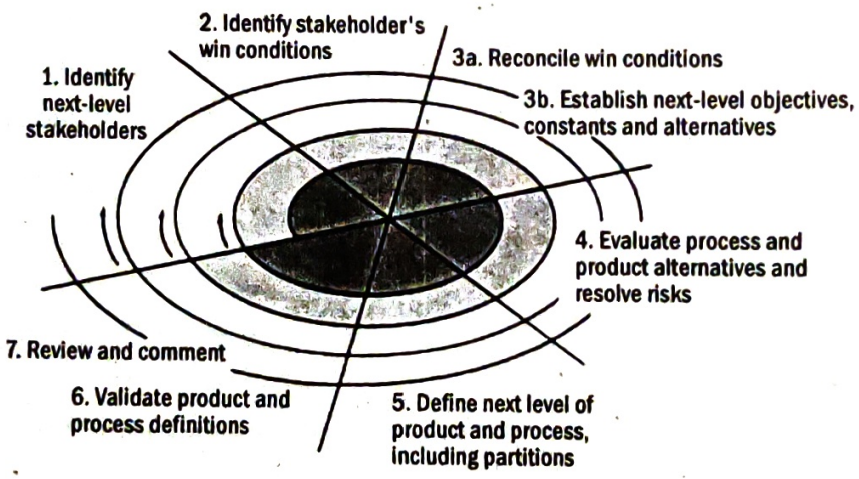


Fig : The WINWIN Spiral Model

- Boehm's WINWIN spiral model defines a set of negotiation activities at the beginning of each pass around the spiral. Rather than a single customer communication activity, the following activities are defined :

(1) Identification of the system or subsystem's key "stakeholders."

(2) Determination of the stakeholders' "win conditions."

(3) Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned (including the software project team).

- Successful completion of these initial steps achieves a win-win results, which becomes the key criterion for proceeding to software and system definition.
- In addition to the emphasis placed on early negotiation, the win-win spiral model introduces three process milestones, called anchor points that helps to establish the completion of one cycle around the spiral and provide decision milestones before the software project proceeds.
- In essence, the anchor points represent three different views of progress as the project traverses the spiral.
- The first anchor point, life cycle objectives (LCO), define a set of objectives for each major software engineering activity.

  Example : As part of LCO, a set of objectives established the definition of top-level system / product requirements.
- The second anchor point, life cycle architecture (LCA), establishes objectives that must be met as the system and software architecture is defined.

  Example : As part of LCA, the software project team must demonstrate that it has evaluated the applicability of off-the-shelf and reusable software components and considered their impact on architectural decisions.
- Initial operational capability (IOC) is the third anchor point and represents a set of objectives associated with the preparation of the software for installation / distribution, site preparation prior to installation and assistance required by all parties that will use or support the software.

Q.30. State the advantage and disadvantages of WIN-WIN spiral model. *CT : W-13(2M)*

Ans. Advantages :

(1) Time and cost saver than spiral model.

(2) Faster software production facilitated through collaborative involvement of the relevant stake holders.

(3) Cheaper software via rework and maintenance reductions.

(4) Better software via use of architectural-level quality - attribute trade off models and early exploration of many architecture options.

Disadvantage :

(1) Alternative location of network or error handling may be compact.

Q.31. What is the significance of the spiral diagram for the UI design process?

Ans.

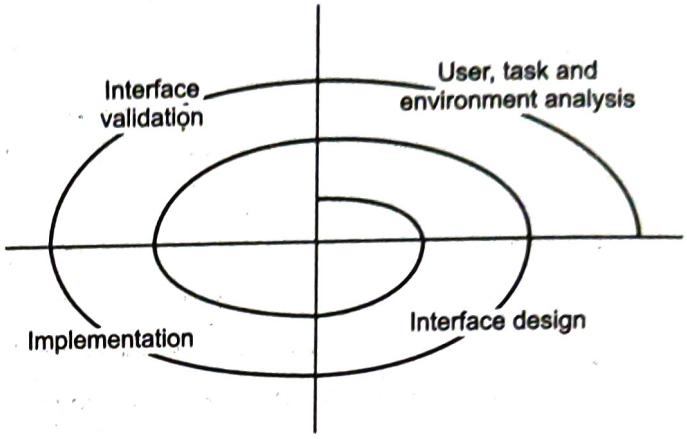


Fig: The user interface design process

The user interface design process encompasses four distinct framework activities :

(1) User, task and environment analysis and modelling.

(2) Interface design.

(3) Interface construction.

(4) Interface validation.

- The initial analysis activity focuses on the profile of the users who will interact with the system. Skill level, business understanding, and general receptiveness to the new system are recorded; and different user categories are defined.
- Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described, and elaborated
- The implementation activity normally begins with the creation of a prototype that enables usage scenarios to be evaluated. As the iterative design process continues, a user interface tool kit may be used to complete the construction of the interface.
- Validation focuses on the ability of the interface to implement every user task correctly, to accommodate all task variations and to achieve all general user requirements.

- Validation focuses on the degree to which the interface is easy to use and easy to learn.
- Validation focuses on the users' acceptance of the interface as a useful tool in their work.

Q.32. Compare waterfall model, spiral model and prototype model.

CT : S-10(6M), CS : W-11(2M)

Ans.

Sr. No.	Features	Original waterfall	Iterative waterfall	Prototyping model	Spiral model
(1)	Requirement Specification.	Beginning	Beginning	Frequently Changed	Beginning
(2)	Understanding Requirements	Well understood	Not Well understood	Not Well understood	Well understood
(3)	Cost	Low	Low	High	Expensive
(4)	Availability of reuseable component	No	Yes	Yes	Yes
(5)	Complexity of system	Simple	Simple	Complex	Complex
(6)	Risk Analysis	Only at beginning	No Risk Analysis	No Risk Analysis	Yes
(7)	User involvement in all phases of SDLC	Only at beginning	Intermediate	High	High
(8)	Gurantee of success	Less	High	Good	High
(9)	Overlapping Phases	No over-lapping	No over-lapping	Yes over-lapping	Yes Over-lapping
(10)	Implemen-tation time	Long	Less	Less	Depends on project
(11)	Flexibility	Rigid	Less Flexible	Highly Flexible	Flexible
(12)	Changes incorporated	Difficult	Easy	Easy	Easy
(13)	Expertise required	High	High	Medium	High
(14)	Cost control	Yes	No	No	Yes
(15)	Resource control	Yes	Yes	No	No

Q.33. Explain the concurrent development model with diagram.

Ans. Concurrent development model :

- The concurrent development model is sometimes called concurrent engineering. Project managers who track project status in terms of the major phases have no idea of the status of their projects.
- The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states. For example, the engineering activity defined for the spiral model is accomplished by invoking the following tasks: prototyping and/or analysis modeling, requirements specification, and design.
- All activities exist concurrently but reside in different states. For example, early in a project the customer communication activity has completed its first iteration and exists in the awaiting changes state.
- The analysis activity now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the analysis activity moves from the under development state into the awaiting changes state.
- The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities. For example, during early stages of design, an inconsistency in the analysis model is uncovered.
- This generates the event analysis model correction which will trigger the analysis activity from the done state into the awaiting changes state.

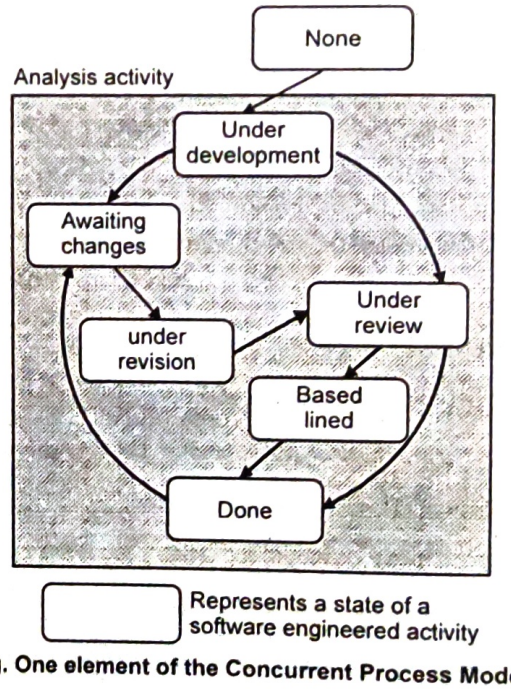


Fig. One element of the Concurrent Process Model

- Concurrency is achieved in two ways as follows :

(1) System and component activities occur simultaneously and can be modelled using the state-oriented approach described previously.

(2) A typical client/server application is implemented with many components, each of which can be designed and realized concurrently.

Q.34. Compare iterative enhancement model and evolutionary developed model

Ans.

Sr. No.	Iterative Enhancement Model	Evolutionary Development Model
(1)	This model has the similar phases as the waterfall model, but with fewer restrictions.	Evolutionary development model bear a resemblance to iterative enhancement model.
(2)	In general the phases occur in the same order as in the waterfall model but these may be conducted in several cycles.	The similar phases as defined for the waterfall model occur here in a cyclical fashion. This model is different from iterative enhancement model in the sense that this doesn't require a useable product at the end of each cycle.
(3)	A utilizable product is released at the end of the each cycle with each release providing additional functionality.	In evolutionary development requirements are implemented by category rather than by priority.

Q.35. State the advantages and disadvantages of concurrent development model.

Ans. Advantages :

(1) The concurrent development model, some times called concurrent engineering. It can be represented schematically as a series of frame work activities, software engineering actions, software engineering task and their associated states.

(2) The concurrent process model defines a series of events that will trigger transition from state to state for each of the software engineering activities and action or task.

(3) The concurrent process model is applicable to all types of software development and provides an accurate picture of the current state of a project.

Disadvantages :

(1) The SRS must be continually updated to reflect changes.

(2) It requires discipline to avoid adding too many new features too late in the project.

SPECIALIZED PROCESS MODELS

Q.36. Explain component based development (CBD) with diagram. *CT : S-08(2M)*

Ans. Component based development :

- The component based development (CBD) model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software.

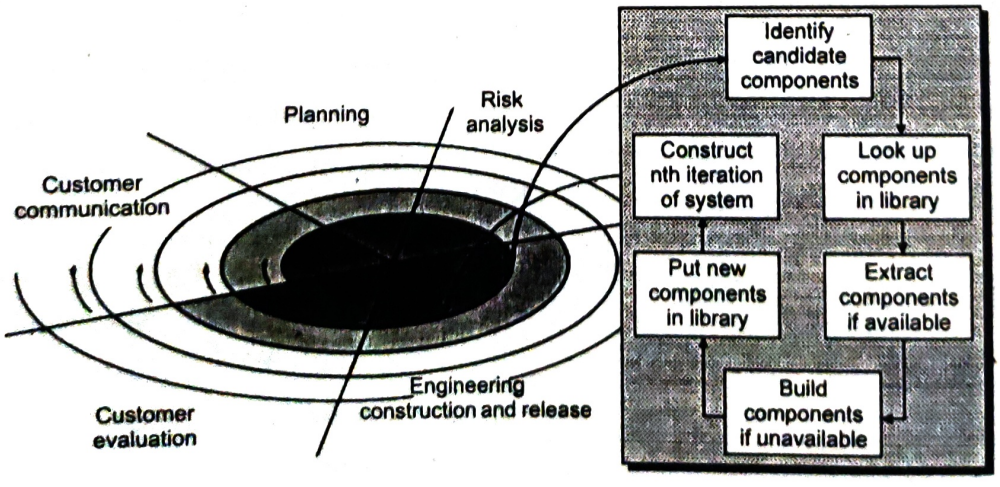


Fig: Component based development

- Software is given to end users for beta testing and user feedback reports both defects and necessary changes. On completion of this phase, we should have a documented software system that is working correctly in its operational environment.

(5) **Production :**

- The production phase of the UP coincides with the deployment activity of the generic process.
- During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.

Q.41. State the advantages and disadvantages of unified process model.

Ans. Advantages :

(1) This is a complete methodology in itself with an emphasis on accurate documentation.

(2) It is proactively able to resolve the project risks associated with the client's evolving requirements requiring careful change request management.

Disadvantages :

(1) Each phase has a milestone which needs to be satisfied for the next particular phase to start. If the respective milestone of the particular phase is not satisfied the entire project might get cancelled or re-engineered before processing further.

(2) The satisfaction criteria of a particular milestone has its own constraints and are not listed specifically.

AGILE PROCESS MODELS

Q.42. What is agile process? What are the various principles of agile process? *CS : S-14(7M)*

Ans. Agile process :

- An agile team is able to respond to changes during project development.
- Agile development recognizes that project plans must be flexible.
- Agility encourages team structures and attitudes that make communication among developers and customers more facile.
- Agility eliminates the separation between customers and developers.
- Agility emphasizes the importance of rapid delivery of operational software and de-emphasizes importance of intermediate work products.
- Agility can be applied to any software process as long as the project team is allowed to streamline tasks and conduct planning in way that eliminate non-essential work products.
- Agile processes based on three key assumptions :

(1) It is difficult to predict in advance which requirements or customer priorities will change and which will not.

(2) For many types of software design and construction activities are interleaved (construction is used to prove the design).

(3) Analysis, design and testing are not as predictable from a planning perspective.

- Agile processes must be adapted incrementally to manage unpredictability.
- Incremental adaptation requires customer feedback based on evaluation of delivered software increments (executable prototypes) over short time periods.

Q.43. State the advantages and disadvantages of agile model.

Ans. Advantages :

(1) Customer satisfaction by rapid, continuous delivery of useful software.

(2) People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.

(3) Working software is delivered frequently (weeks rather than months).

(4) Face-to-face conversation is the best form of communication.

(5) Close, daily cooperation between business people and developers.

(6) Continuous attention to technical excellence and good design.

(7) Regular adaptation to changing circumstances.

(8) Even late changes in requirements are welcomed.

Disadvantages :

(1) In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

(2) There is lack of emphasis on necessary designing and documentation.

(3) The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

(4) Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

Q.44. What are different agile process models? Explain each in detail.

OR Explain Adaptive Software Development process in brief.

CS : W-13(5M)

OR Explain the extreme programming process with a neat diagram.

CS : S-10(6M)

Ans. Different agile process models are as follows :

(1) Extreme Programming (XP)

(2) Adaptive Software Development (ASD)

(3) Dynamic Systems Development Method (DSDM)

(4) Scrum

(5) Crystal

(6) Feature Driven Development (FDD)

(7) Agile Modeling (AM)

(1) **Extreme Programming :** Extreme programming involves a number of practices :

- Incremental development is supported through small, frequent releases of the system. Requirements are based on simple customer stories or scenarios that are used as a basis for deciding what functionality should be included in a system increment.
- Customer involvement is supported through the continuous engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
- People, not process, are supported through pair programming, collective ownership of the system code and a sustainable development process that does not involve excessively long working hours.
- Change is embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration and continuous integration of new functionality.
- Maintaining simplicity is supported by constant refactoring that improves code quality and by using simple designs that do not unnecessarily anticipate future changes to the system.

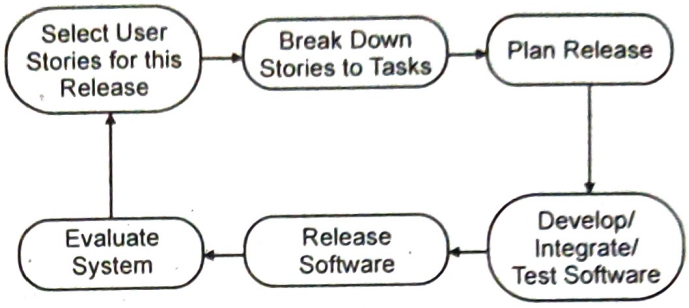


Fig.(a) The extreme programming release cycle

(2) **Adaptive Software Development :**

- Self-organization arises when independent agents cooperate to create a solution to a problem that is beyond the capability of any individual agent.
- Emphasizes self-organizing teams, interpersonal collaboration and both individual and team learning.
- Adaptive cycle characteristics Phases are as follows :

(i) Mission-driven.

(ii) Component-based.

(iii) Iterative.

(iv) Time-boxed.

(v) Risk driven and change-tolerant.

(vi) Speculation (project initiated and adaptive cycle planning takes place).

(vii) Collaboration (requires teamwork from a jelled team, joint application development is preferred requirements gathering approach, minispecs created).

(viii) Learning (components implemented and tests, focus groups provide feedback, formal technical reviews, postmortems).

(3) **Dynamic Systems Development Method :**

- It provides a framework for building and maintaining systems which meet tight time constraints using incremental prototyping in a controlled environment.

(2) **Principle 2 : Assemble and test complete delivery package :**

- It is not the case that the deliverable part is 'only software'. The customer must get all support and essential help from developer's side. Hence, a CD with complete assembled and tested delivery package with following support should be delivered :

(i) All the executable files of the developed software.

(ii) All the installation procedures.

(iii) All the supported operational features and help.

(iv) The essential manuals required for software trouble-shooting.

(v) Information of all operational requirements of the developed software.

- For example, hardware, operating system, networking components etc.

(3) **Principle 3 : A support regime must be established before the software is delivered :**

- An end user expects responsiveness and accurate information when a question or problem arises.
- Support should be planned, support materials should be prepared and appropriate record keeping mechanisms should be established so that the software team can conduct a categorical assessment of the kinds of support requested.

(4) **Principle 4 : Appropriate instructional materials must be provided to end users :**

Appropriate training aids (if required) should be developed, troubleshooting guidelines should be provided.

(5) **Principle 5 : Buggy software should be fixed first, delivered later:**

Because of under time pressure, some software organizations deliver low-quality increments with a warning to the customer that bugs "will be fixed in the next release."

SYSTEM ENGINEERING HIERARCHY

Q.14. Write a short note on Software Engineering and System engineering.

Ans. **Software engineering :**

- Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.
- Software engineering is performed by creative, knowledgeable people who should work within a defined and mature software process.
- Software engineering is an outgrowth of hardware and system engineering. It encompasses a set of three key elements methods, tools, and procedures-that enable the manager to control the process of software development and provide the practitioner with a foundation for building high-quality software in a productive manner.

System engineering :

- A process that is concerned with specifying a system, integrating its components and testing that the system meets its requirements.
- The system engineering process is called business process engineering when the context of the engineering work focuses on a business enterprise.
- Software engineering occurs as a consequence of a process called system engineering.
- System engineering focuses on a variety of elements, analyzing, designing and organizing those elements into a system that can be a product, a service, or a technology for the transformation of information or control.
- A product includes everything from a wireless telephone to an air traffic control system is to be built, the process is called product engineering.
- Both business process engineering and product engineering attempt to bring order to the development of computer-based systems.

Q.15. Differentiate between software engineering and system engineering. *CT: S-09(3M)*

Ans.

Sr. No.	Software engineering	System engineering
(1)	Software engineering is the study and application of engineering to the design, development and maintenance of software.	Systems engineering is an interdisciplinary field of engineering that focuses on how to design and manage complex engineering projects over their life cycles.

Sr. No.	Software engineering	System engineering
(2)	Software engineering is a "component" in the process of system engineering.	System engineering is concerned with all aspects of computer based systems development including hardware, software and process engineering.
(3)	Software engineering is to tell the practicalities of developing and delivering useful software.	System engineering is to identify the roles of hardware, software, people, database and other system elements involved with that system which is going to be developed.
(4)	Software engineering focus more on implementing quality software.	System engineers focus more on users and domains.
(5)	Software engineers will focus solely on software components.	System engineers may deal with a substantial amount of hardware engineering.
(6)	Software engineering methods adapted to system engineering includes : • Model-Driven Development. • UML-SysML • Use cases • Object-oriented Design. • Iterative development. • Agile methods • Continuous integration • Process modeling • Process improvement • Incremental V & V	System engineering methods adapted to software engineering includes : • Stakeholder analysis. • Requirements engineering. • Functional decomposition. • Design constraints • Architectural design. • Design tradeoffs • Design criteria • Interface specification • Traceability • Configuration management • Systematic verification and validation.

Q.16. Write and explain the characteristics of KPA (key process area).

CT: S-06 (4M), W-07(3M)

OR Explain five process maturity levels (CMM) and also list out the KPA associated with each process maturity levels.

CT: S-10(8M), W-13(7M)

OR What is Process Key Identifier (PKI) ? Explain various level at SEI-CMM model with respect to the PKI followed in that level.

CT: S-14(7M)

OR What is KPAs ? Explain.

Ans. KPA :

- KPA stand for key process areas. The KPAs describe those software engineering functions that must be present to satisfy good practice at a particular level.
- Each KPA is described by identifying the following characteristics:

(i) **Goals :** The overall objectives that the KPA must achieve.

(ii) **Commitments :** Requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.

(iii) **Abilities :** Those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.

(iv) **Activities :** The specific tasks required to achieve the KPA function.

(v) **Methods for monitoring implementation :** The manner in which the activities are monitored as they are put into place.

(vi) **Methods for verifying implementation :** The manner in which proper practice for the KPA can be verified.

- Eighteen KPAs are defined across the maturity model and mapped into different levels of process maturity.
- The following KPAs should be achieved at each process maturity level:

(1) **Process maturity level 2 :**

- Software configuration management.
- Software quality assurance.
- Software subcontract management.
- Software project tracking and oversight.
- Software project planning.
- Requirements management.

(2) **Process maturity level 3 :**

- Peer reviews.
- Intergroup coordination.
- Software product engineering.
- Integrated software management.

Q.36. Explain the FAST guidelines principles for requirement engineering suggested by Davis.

Ans. Davis suggests a set of guiding principles for requirements engineering are as follows:

(1) Understand the problem before we begin to create the analysis model. There is a tendency to rush to a solution, even before the problem is understood. This often leads to elegant software that solves the wrong problem.

(2) Develop prototypes that enable a user to understand how human/machine interaction will occur.

(3) Since the perception of the quality of software is often based on the perception of the "friendliness" of the interface, prototyping are highly recommended.

(4) Record the origin of and the reason for every requirement. This is the first step in establishing traceability back to the customer.

(5) Use multiple views of requirements. Building data, functional and behavioural models provide the software engineer with three different views.

(6) Tight deadlines may preclude the implementation of every software requirement. If an incremental process model is applied, those requirements to be delivered in the first increment must be identified.

(7) Work to eliminate ambiguity. Because most requirements are described in a natural language, the opportunity for ambiguity abounds. The use of formal technical reviews is one way to uncover and eliminate ambiguity.

POINTS TO REMEMBER :

(1) A system engineering focuses on a variety of elements, analysing, designing and organizing those elements into a system that can be product, service or technology for transformation of information or control.

(2) KPA stands for key process area which describes those software engineering functions that must be present to satisfy good practice at a particular level.

(3) Computer based system is a set of arrangement of elements that are organized to accomplish some predefined goal by processing information.

(4) Business process engineering is one approach for creating an overall plan for implementing the computing architecture.

(5) Product engineering is translation of customer's desire for a set of defined capabilities into a working product.

(6) System engineering is considered as a modeling process.

(7) Software requirement engineering is a process in which the system requirements are discovered, refined modelled and specified.

(8) The requirements engineering process can be described in five distinct steps :

(i) Requirements elicitation.

(ii) Requirements analysis and negotiation.

(iii) Requirements specification.

(iv) System modeling.

(v) Requirements validation.

(9) Software requirement specification is produced in the phase of analysis task which sets the targets and objective of software.

(10) Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.

(11) Jackson system development is a method of system development that covers the software life cycle, either directly or by providing a framework into which more specialized techniques can adjust.

(12) A concept called feasibility study is an effective way to safeguard against excessive consumption of resources.

(13) In system specification, all information required for a description of software scope is available and documented before software project planning begins.

(14) A team-oriented approach for requirement elicitation is applied during early steps of analysis and specification called Facilitated Application Specification Techniques (FAST).

(1) Separate functionality from implementation.

(2) Develop a model of the desired behaviour of a system that encompasses data and the functional responses of a system to various stimuli from the environment.

(3) Establish the context in which software operates by specifying the manner in which other system components interact with software.

(4) Define the environment in which the system operates and indicate how "a highly intertwined collection of agents react to stimuli in the environment (changes to objects) produced by those agents".

(5) Create a cognitive model rather than a design or implementation model. The cognitive model describes a system as perceived by its user community.

(6) A specification is always a model an abstraction of some real situation that is normally quite complex. Hence, it will be incomplete and will exist at many levels of detail.

(7) Establish the content and structure of a specification in a way that will enable it to be amenable to change.

- The list of basic specification principles provides a basis for representing software requirements.
- However, principles must be translated into realization.

Q.35. Explain the FAST approach for gathering the requirement.

OR Explain 'FAST' method of requirement gathering.

CT: W-11(6M)

OR Explain Facilitated Applications Specifications Techniques (FAST) in detail. *CT: W-12(6M), CS: S-13(6M)*

Ans. FAST approach :

- In many cases customers and software engineers often have an unconscious "us and them" mindset.
- Other than working as a team to identify and refine requirements, each constituency defines its own "territory" and communicates through a series of memos, formal position papers, documents and question and answer sessions.
- Misunderstandings abound, important information is omitted, and a successful working relationship is never established.
- With these problems in mind, a number of independent investigators have developed a team-oriented approach to requirements gathering that can be applied to help establish the scope of a project, called facilitated application specification techniques (FAST), this approach encourages the creation of a joint team of customers and developers who work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of requirements.
- The following are the basic guidelines to FAST :

(i) A meeting is held at a neutral site and participated by both the developer and the customer.

(ii) Rules are already made for preparation and participation.

(iii) An agenda is suggested that covers all important points but at the same time encourages the free flow of ideas.

(iv) A facilitator controls the meeting. He can be a customer, a developer or an outsider.

(v) A definition mechanism is used. It can be work sheets, flip charts, or wall stickers or it may be an electric bulletin board, chat room or virtual forum.

(vi) The goal is problem identification, proposing solution elements, different approaches negotiation and specifying a preliminary set of solution requirements in such an environment that is conductive for the goal accomplishment.

- Starting with the initial meetings between the customer and developer, the scope of the problem and overall perception of a solution is established.
- As a result of these meetings, the customer and developer write a one or two page product request.
- Then decides the meeting place, time and date for FAST and a facilitator is chosen.
- Attendees from both the sides are invited.
- The producer request is distributed to all of them before the day of meeting.
- During request review (in the days before the meeting), each attendee is asked to write objects that are part of the environment surrounding the system, other objects to be produced by the system and the third set of objects that are used by the system for performing its functions.
- Moreover, each one is asked to prepare another list of services, i.e. processes or functions, manipulating or interacting with the objects.
- At last, constraints list and performance criteria list are also prepared.
- The lists are not expected to be exhaustive but are expected to represent each person's opinion about the system.

(3) **Systems design :** A general design is developed with the purpose of planning for the construction of the new system.

(4) **Systems development :** The new system is created.

(5) **System installation :** The current operation is converted to run on the new system.

(6) **Systems evaluation and monitoring :** The newly operational system is evaluated and monitored for the purpose of enhancing its performance and adding value to its functions.

(7) Looping back from a later phase to an earlier one may occur if the need arises.

Feasibility study :

- The feasibility study investigates the problem and the information needs of the stakeholders.
- It seeks to determine the resources required to provide an information systems solution, the cost and benefits of such a solution, and the feasibility of such a solution. The analyst conducting the study gathers information using a variety of methods are:

(i) Interviewing users, employees, managers, and customers.

(ii) Developing and administering questionnaires to interested stakeholders, such as potential users of the information system.

(iii) Observing or monitoring users of the current system to determine their needs as well as their satisfaction and dissatisfaction with the current system.

(iv) Collecting, examining, and analyzing documents, reports, layouts, procedures, manuals, and any other documentation relating to the operations of the current system.

(v) Modeling, observing, and simulating the work activities of the current system.

Economic feasibility :

- The economic viability of the proposed system. The proposed project's costs and benefits are evaluated.
- Tangible costs include fixed and variable costs, while tangible benefits include cost savings, increased revenue, and increased profit.
- A project is approved only if it covers its cost in a given period of time. However, a project may be approved only on its intangible benefits such as those relating to government regulations, the image of the organization, or similar considerations.

Technical feasibility :

- The possibility that the organization has or can procure the necessary resources.
- This is demonstrated if the needed hardware and software are available in the marketplace or can be developed by the time of implementation.

Operational feasibility :

- The ability, desire, and willingness of the stakeholders to use, support, and operate the proposed computer information system.
- The stakeholders include management, employees, customers, and suppliers.
- The stakeholders are interested in systems that are easy to operate, make few, if any, errors, produce the desired information, and fall within the objectives of the organization.

Q.9. Write a short note on quality function deployment (QFD).

CT: W-09(5M), W-10(6M)

OR What do you mean by Quality Function Deployment (QFD) ? Also enlist the three types of requirements identified by QFD.

CT: S-11(3M), CS: S-10(4M), W-10(6M),

CS: W-12(4M),W-13(3M)

Ans. Quality function deployment (QFD) :

- Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
- In the early 1970s, QFD concentrates on maximizing customer satisfaction from the software engineering process.
- To accomplish this, QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.
- QFD identifies three types of requirements :

(1) **Normal requirements :**

- The objectives and goals that are stated for a product or system during meetings with the customer.
- If these requirements are present, the customer is satisfied. Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance.

(2) **Expected requirements :**

- These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them.
- Their absence will be a cause for significant dissatisfaction.
- Examples of expected requirements are ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation.

(3) **Exciting requirements :**

- These features go beyond the customer's expectations and prove to be very satisfying when present. For example, word processing software is requested with standard features.
- The delivered product contains a number of page layout capabilities that are quite pleasing and unexpected.

Q.10. What is software prototyping? Explain different steps involved in software prototyping. ***CT: W-08,09(6M), S-13(7M)***

OR How software prototyping is useful in requirement gathering? Explain in detail. ***CT: W-11(6M)***

OR Explain software protyping and also explain the different approaches for software prototyping. ***CT: S-12(7M)***

OR Write short note on different prototyping methods and tools. ***CS: W-08(2M), W-09(3M)***

Ans. Software prototyping :

- Software prototyping is the process of building a model of the software to be built, called a prototype, for customer and developer assess-ment.
- Some circumstances require the construction of a prototype at the initial stages of analysis, since the model is the only means by which requirements can be derived effectively. This model then evolves into production software.
- The prototyping approach, often called throw-away prototyping, helps a prototype driving solely as a rough demonstration of requirements. It is then discarded, but a different approach is used for software development.
- An open-ended approach, on the other hand, is called an evolutionary prototyping. It used the prototype in the initial stages of the analysis activity that will be continued into design and construction. The prototype of the software is first evolution of the final product.
- Before choosing an approach, it is must to determine whether the system to be built is amenable to prototyping.
- Following are the prototyping candidacy factors :

(i) Application area.

(ii) Application complexity.

(iii) Customer characteristics.

(iv) Project characteristics.

- Generally, any application that can do any of the following is a candidate for prototyping :

(i) Creates dynamic visual displays.

(ii) Interacts much with a user.

(iii) Demands algorithms or combinational processing that must be developed in an evolutionary fashion.

- However, the above noted applications must be weighed against application complexity. If a candidate application (having the above characteristics) will demand the development of several lines code before performing any demonstrable function, it may be too complex for prototyping.
- Since in later steps, the customer must interact with the prototype, the following are essential :

(i) Customer resources must be fully committed to prototype evaluation and its refinement.

(ii) The customer should be capable of making requirements decisions in short intervals of time.

- At last, the project nature will have a strong bearing on the prototyping efficacy. Andriole suggests six questions, given in table below, and indicate typical sets of answers and the corresponding suggested prototyping approach.

Selecting the Appropriate prototyping approach :

Sr. No.	Questions	Throwaway Prototype	Evolutionary prototype	Additional Preliminary word required
(1)	Is the application domain understood?	Yes	Yes	No
(2)	Can the problem be modeled ?	Yes	Yes	No

(i) User, task, and environment analysis and modelling.

(ii) Interface design.

(iii) Interface construction.

(iv) Interface validation.

- The spiral shown in above figure implies that each of these tasks will occur more than once, with each pass around the spiral representing additional elaboration of requirements and the resultant design.
- The analysis activity focuses on the profile of the users who will interact with the system. Skill level, business understanding, and general receptiveness to the new system are recorded and different user categories are defined.
- Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described, and elaborated (over a number of iterative passes through the spiral).
- The implementation activity normally begins with the creation of a prototype that enables usage scenarios to be evaluated.
- Validation focuses on:

(i) The ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements.

(ii) The degree to which the interface is easy to use and easy to learn.

(iii) The users' acceptance of the interface as a useful tool in their work.

Q.16. Write a note on modeling the system architecture.

CT:W-07(3M), W-08(6M), CS:W-10(7M)

Ans. Modeling the system architecture :

- Systems architecture is a generic discipline to handle objects called "systems", in a way that supports reasoning about the structural properties of these objects.
- Systems architecture is a response to the conceptual and practical difficulties of the description and the design of complex systems.
- A system architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them.
- It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system.
- A system architecture primarily concentrates on the internal interfaces among the system's components or subsystems, and on the interface(s) between the system and its external environment, especially the user.

OBJECT ORIENTED ANALYSIS

Q.17. What is object oriented analysis?

OR Discuss the importance of object oriented analysis.

OR Explain the steps in object oriented analysis.

CT : W-06,08,09, S-11(7M)

Ans. Object oriented analysis :

- The objective of object-oriented analysis (OOA) is to develop a series of models that describe computer software as it work to satisfy a set of customer defined requirements.
- OOA is grounded in a set of basic principles in order to build an analysis model.
- There are five basic principles as applied :

(i) The information domain is modeled.

(ii) Function is described.

(iii) Behaviour is represented.

(iv) Data, functional and behavioural models are partitioned to expose greater detail.

(v) Early model represent the essence of the problem while later models provide implementation details.

- Basic aim of Object Oriented Analysis (OOA) is to define all classes (also relationship and behaviour associated with t hem) those are required to solve problem under consideration.
- To complete this tasks, following tasks are carried out :

(i) The user requirements are finalized by proper communication between customer and developer.

(ii) The classes are defined from given problem statement. (i.e. attributes and methods are defined).

(iii) A class hierarchy is defined. That means the classes, subclasses and their relation hierarchy is defined.

(iv) Object oriented relationship i.e. object connections are established.

(v) Object behaviour model is created.

- Above 5 tasks are repeated in itervative way until the model is completed.

Q.18. State the difference between an object used in OOA and object used during data modeling. *CT : W-07(6M)*

Ans. OOA :

- To perform object-oriented analysis, a software engineer should perform the following generic steps :

(i) Elicit customer requirements for the system.

(ii) Identify scenarios or use-cases.

(iii) Select classes and objects using basic requirements as a guide.

(iv) Identify attributes and operations for each system object.

(v) Define structures and hierarchies that organize classes.

(vi) Build an object-relationship model.

(vii) Build an object-behavior model.

(viii) Review the OO analysis model against use-cases or scenarios.

OOD :

- The terminology and process steps for each of these OOD methods differ, the overall OOD processes are reasonably consistent.
- To perform object-oriented design, a software engineer should perform the following generic steps:

(i) Describe each subsystem and allocate it to processors and tasks.

(ii) Choose a design strategy for implementing data management, interface support, and task management.

(iii) Design an appropriate control mechanism for the system.

(iv) Perform object design by creating a procedural representation for each operation and data structures for class attributes.

(v) Perform message design using collaborations between objects and object relationships.

(vi) Create the messaging model.

(vii) Review the design model and iterate as required.

Q.19. Explain booch method for object modeling. *CT : S-12(7M)*

OR Explain five methods of object oriented analysis. *CS : W-08(10M)*

OR Explain the rumbaugh method for object modeling. *CT : S-09(7M), CS : S-09(7M)*

OR Explain in detail about design techniques for object oriented software engineering. *CT : S-14(7M)*

Ans. The methods of object oriented analysis are :

(1) Booch method.

(2) Rumbaugh method.

(3) Jacobson method.

(4) Coad and vourdon method.

(5) Wirfs-Brock method.

(1) Booch method :

- The Booch methd extensively uses diagramming techniques to develop the system.
- It uses the following diagrams :

  (i) Class diagrams

  (ii) State transition

  (iii) Object diagrams

  (iv) Interaction diagrams
- The methodology operates through a two-stage development, one macro and the other micro.
- Macro processes deals with conceptualization, analysis and development of the model, designing system architecture, implementation and maintenance.
- The micro development process stage consists of identifying four items class and objects, its semantics, its relationship and interfaces and implementation.
- The macro process controls the micro process.

(2) Rumbaugh method :

- Rumbaugh evolved the object oriented modeling technique (OMT), which the describes a method of analysis, system design, object design and implementation of the system.
- OMT recommends use of these four phases iteratively.
- Table shows the phases and the outcome of each phase:

Table : OMT phases and outcomes

Phase	Outcome
OO analysis	Classes and system models using the objects.
OO system design	System architecture using classes, objects.
Object design document	Design document detailing static and dynamic objects and functional models.
Implementation	Coding, testing and deployment.

- The OMT methodology, using three models object, dynamic and functional, is one of the strongest approaches to analysis and design of a software system.
- The object model describes the structure of the objects in the system in terms of Identity, relationship to other objects, attributes and operations.

- A continuous process that begins with the assessment of a customer specification and ends with design is proposed.
- A brief outline of Wirfs-Brock analysis-related tasks follows:

(i) Evaluate the customer specification.

(ii) Extract candidate classes from the specification via grammatical parsing.

(iii) Group classes in an attempt to identify super classes.

(iv) Define responsibilities for each class.

(v) Assign responsibilities to each class.

(vi) Identify relationships between classes.

(vii) Define collaboration between classes based on responsibilities.

(viii) Build hierarchical representations of classes.

(ix) Construct a collaboration graph for the system.

Q.20. Define the terms :

(a) Class *CS:W-08(3M)*

(b) Inheritance and encapsulation. *CT:W-09(3M), S-11(6M)*

(c) Polymorphism with example.

OR Explain different components of object oriented analysis.

CT : S-07(3M)

Ans.

(a) Class :

- A class is an OO concept that encapsulates the data and procedural abstractions required to describe the content and behaviour of some real world entity.

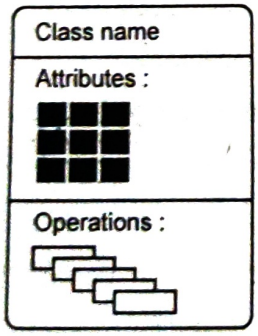


Fig. Class

- All objects that exist within a class inherit its attributes and the operations that are available to manipulate the attributes.
- A superclass is a collection of classes, and a subclass is a specialized instance of a class.
- These definitions imply the existence of a class hierarchy in which the attributes and operations of the superclass are inherited by subclasses that may each add additional "private" attributes and methods.

(b) Inheritance :

- Inheritance is a mechanism that enables the responsibilities of one object to be propagated to other objects.
- Inheritance occurs throughout all levels of a class hierarchy. Because inheritance is a pivotal characteristic in many OO systems, many OO metrics focus on it.
- Examples include number of children (number of immediate instances of a class), number of parents (number of immediate generalizations), and class hierarchy nesting level.

(c) Encapsulation :

- The Object Oriented class and the objects spawned from the class encapsulate data and the operations that work on the data in a single package.
- The internal implementation details of data and procedures are hidden from the outside world (Data hiding).
- This reduces the propagation of side effects when changes occur.
- Interfaces among encapsulated objects are simplified. An object that sends a message need not be concerned with the details of internal data structures.
- Hence, interfacing is simplified and the system coupling tends to be reduced.

(d) Polymorphism :

- Polymorphism is a characteristic that greatly reduces the effort required to extend an existing OO system.
- To understand polymorphism, consider a conventional application that must draw four different types of graphs: line graphs, pie charts, histograms, and Kiviat diagrams.
- Ideally, once data are collected for a particular type of graph, the graph should draw itself.
- To accomplish this in a conventional application, it would be necessary to develop drawing modules for each type of graph.
- Polymorphism enables a number of different operations to have the same name, reducing the number of lines of code required to implement a system and facilitating changes when they are made.

```
class Calculation
{
void sum(int a,int b)
{
System.out.println(a+b);
}
void sum(int a,int b,int c)
{
System.out.println(a+b+c);
}
public static void main(String args[])
{
Calculation obj=new Calculation();
obj.sum(10,10,10);
obj.sum(20,20);
}
}
```

Q.27. What is the difference between class variable and object?

CT:W-06(3M), W-08(4M)

Ans.

Sr. No.	Class	Objects
(1)	A class is an OO concept that encapsulates the data and procedural abstractions required to describe the content and behavior of some real world entity.	Objects are identified by examining the problem statement or by performing a "grammatical parse" on the processing narrative for the system to be built.
(2)	A superclass is a collection of classes, and a subclass is a specialized instance of a class	All objects that exist within a class inherit its attributes and the operations that are available to manipulate the attributes.
(3)	These definitions imply the existence of a class hierarchy in which the attributes and operations of the superclass are inherited by subclasses that may each add additional "private" attributes and methods.	Objects are determined by underlining each noun or noun clause and entering it in a simple table.

Q.28. How does one know whether a potential object is a good candidate for use is an object oriented system?

OR Explain how objects and classes are identified in object oriented analysis. **CT: S-06, W-10(6M)**

Ans.

- Objects are identified by examining the problem statement or by performing a "grammatical parse" on the processing narrative for the system to be built. Objects are determined by underlining each noun or noun clause and entering it in a simple table.
- The following six selection characteristics for good candidate of potential object for use is an object oriented system :

(1) **Retained information:** The potential object will be useful during analysis only if information about it must be remembered so that the system can function.

(2) **Needed services :** The potential object must have a set of identifiable operations that can change the value of its attributes in some way.

(3) **Multiple attributes :** During requirement analysis, the focus should be on "major" information; an object with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another object during the analysis activity.

(4) **Common attributes :** A set of attributes can be defined for the potential object and these attributes apply to all occurrences of the object.

(5) **Common operations :** A set of operations can be defined for the potential object and these operations apply to all occurrences of the object.

(6) **Essential requirements :** External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as objects in the requirements model.

SCENARIO BASED MODELING

Q.29. Draw the activity diagram for access camera surveillance-display camera views function. **CS : S-10(8M)**

OR What is the role of use-cases in scenario based modeling?

CS : W-11(6M)

OR Draw and explain activity diagram of elicitation process.

CS : W-12(4M)

- A few simple guidelines can aid immeasurably during the derivation of a data flow diagram :

(i) The level 0 data flow diagram should depict the software/system as a single bubble.

(ii) Primary input and output should be carefully noted.

(iii) Refinement should begin by isolating candidate processes, data objects, and data stores to be represented at the next level.

(iv) All arrows and bubbles should be labeled with meaningful names.

(v) Information flow continuity must be maintained from level to level.

(vi) One bubble at a time should be refined. There is a natural tendency to overcomplicate the data flow diagram.

- A level 0 DFD for the security function is shown in figure.
- The primary external entities produce information for use by the system and consume information generated by the system.
- The labeled arrows represent data objects or data object hierarchies.
- For example, user commands and data encompasses all configuration commands, all activation and deactivation commands, all miscellaneous interactions, and all data that are entered to qualify - or expand a command.

Q.31. Write short note on DFD. ***CT:S-07, W-12(3M), W-10(7M)*** ***CS : W-08(3M), W-10(4M)***

OR Explain data flow diagram (DFD) with all symbols used for a DFD. Draw a DFD for order processing in shop. ***CT : S-10(6M)***

OR What is the importance of DFD? List the notations used to design DFD. ***CS : S-14(2M)***

Ans. Data flow diagram :

- DFD is also called as bubble chart. It is a simple graphical representation that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the data generated by the system, various processing carried out on these data, and the output data generated by the system.
- The DFD technique is popular because of the fact that DFD is a very simple representation, it is simple to understand and use.
- A DFD model uses a very limited number of primitive symbols to represent the function performed by a system and data flow among these function.
- Starting with a set of high level functions i.e. system performs a DFD model hierarchically represents various sub function.

Example : The data flow diagram developed for medicine management system includes the external entities : Whole - seller, customer and user. The DFD shows following requirements with proper flow of data from one entity to other.

- The medicines are purchased from whole seller.
- The customer can request for the medicines in medicine store.
- User can request for display of the current status of the items in medicine.
- When medicines are purchased from whole seller, the system generates the payment for the whole-seller.
- When medicines are sold to the customer, the system generates the bill for the customer.
- The system also generates the required information for the user as per his requirements.

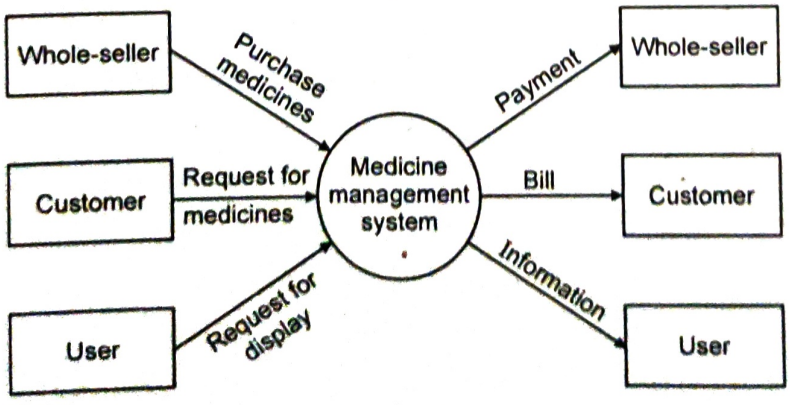


Fig. Data flow diagram for medicine managements system

- The notations of DFD are as follows .

Sr. No.	Symbol	Meaning	Explanation
(1)		Functional symbol	A function is represented using a circle. This symbol is called a process or a bubble. Bubbles are connected with the names of corresponding function.
(2)		External entity symbol	An external entity such as librarian, a library member, etc. is represented by a rectangle. The external entities are essentially those physical entity external to the software system which interact with the system by inputting data to the system or by consuming the data produced by the system.
(3)		Data flow	A directed arc on an 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. Data flow symbols are usually annotated with the corresponding data names.
(4)		Data store symbol	A data store represents a logical file. It is represented using open rectangle lines. A logical file can represent either a data store symbol which can represent either a data structure or a physical file on a list. Each data store is connected to a process by means of a data stored symbol.
(5)		Output symbol	The o/p symbol is used when a hard copy is produced and the user of the copies can not be clearly specified or there are several users of the o/p.

Q.32. List steps for creating data flow diagram.

Ans. Steps for creating data flow diagram are :

(1) Make a list of business activities and use it to determine various external entities data flow processes data stores.

(2) Create a context diagram which shows external entities and data flows to and from the system. Do not show any detailed-processes or data stores.

(3) Draw diagram of the next level. Show processes but keep them general. Show data stores at this level.

(4) Create a child diagram for each of the processes in DFD.

(5) Check for errors and make sure the tables we assign to each process and data flow are meaningful.

(6) Develop physical data flow diagram from the logical data flow diagram. Distinguish between manual and automated processes, describe actual files and report by name, and add controls to indicate when processes are complete or error occur.

(7) Partition the physical data flow diagrams by separation or grouping parts of the diagram in order to facilitate programming and implementation.

Q.33. What is the importance of CFD? List the notations used to design CFD. *CS : S-14(2M)*

OR Write short note on control flow diagram. *CS : W-08(3M)*

Ans. Control flow diagram :

- Out of many type of data processing applications, the data model and the data flow diagram are all that is necessary to obtain a meaningful insight into software requirements.
- The large class of applications having following characteristics requires control flow modeling.

(1) The applications those are driven by the events rather than data.

(2) The application those produce control information rather than reports or displays.

(3) The applications which process information in specific time.

(4) The control item or event is implemented as Boolean value for example, true or false; on or off; 1 or 0

- To create a control flow model, a data flow model is "stripped" of all data flow arrows. Events and control items (dashed arrows) are then added to the diagram and a "window" (a vertical bar) into the control specification.
- An event or control item is implemented as a Boolean value or a discrete list of conditions.
- To select potential candidate events, the following guidelines are suggested :

(i) List all sensors that are "read" by the software.

(ii) List all interrupt conditions.

(iii) List all "switches" that are actuated by an operator.

(iv) List all data conditons.

(v) Recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible CSPEC inputs/outputs.

(vi) Describe the behaviour of the system by identifying its states; identify how each state is reached; and define the transition between states.

(vii) Focus on possible omissions - a very common error in specifying control; for example, ask : "Is there any other way I can get to this state or exit from it?".

A level 1CFD for SafeHome software is shown in Fig.

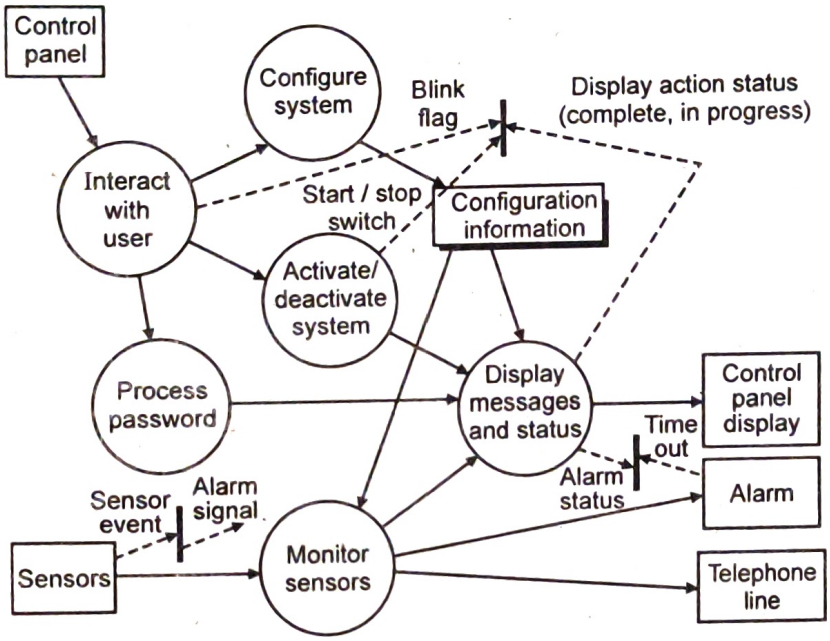


Fig. Level1 CFD for safeHome

- Among the events and control items noted are sensor event (i.e. a sensor has been tripped), blink flag (a signal to blink the LCD display), and start/stop switch (a signal to turn the system on or off).
- When the event flows into the CSPEC window from the outside world, it implies that the CSPEC will activate one or more of processes shown in the CFD.
- When a control item emanates from a process and flows into the CSPEC window, control and activation of some other process or an outside entity is implied.

Q.34. Explain in detail :

(i) Control specification.

(ii) Process specification.

OR Explain control specification diagram and process specification diagram in detail.

Ans.

(i) Control specification :

- The control specification (CSPEC) represents the behaviour of the system (at the level from which it has been referenced) in two different ways.

- In many programming languages, an 'n' dimensional space is called an array.
- Items, a vectors, and spaces may be organized in a variety of formats.
- A linked list is a data structure that organizes non contiguous scalar items, vectors, or spaces in manner (called nodes) that enables them to be processed as a list.
- Each node contains the appropriate data organization (e.g. a vector) and one or more pointers that indicate the address in storage of the next node in the list.
- Nodes may be added at any point in the list by redefining pointers to accommodate the new list entry.
- Other data structures incorporate or are constructed using the fundamental data structures just described.
- For example, a hierarchical data structure is implemented using multi-linked lists the contain scalar items, vectors, and possibly, n-dimensional space.
- A hierarchical structure is commonly encountered in applications that require information category-zation and associativity.
- It is important to note that data structures, like program structure, can be represented at different levels of abstraction.
- For example, a stack is a conceptual model of a data structure that can be implemented as a vector or a linked list.
- Depending on the level of design detail, the internal workings of a stack may or may not be specified.

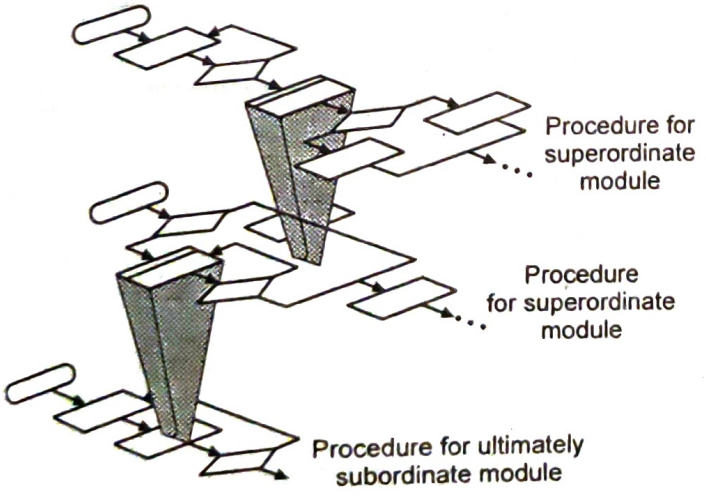


Fig. Procedure is layered

Q.37. Write short note on data dictionary.

CT: W-12(4M), CS : W-08(2M), S-09(3M)

Ans. Data Dictionary :

- 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 inputs, outputs, components of stores and intermediate calculations.
- The analysis model encompasses representations of data objects, function, and control.
- In each representation data objects and/or control items play a role. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data object and control item.
- The data dictionary is always implemented as part of a CASE "structured analysis and design tool." The format of dictionaries varies from tool to tool.
- It contains the following information :

(1) **Name :** The primary name of the data or control item, the data store or an external entity.

(2) **Alias :** Other names used for the first entry.

(3) **Where-used/how-used :** A listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity)

(4) **Content description :** A notation for representing content.

(5) **Supplementary information :** Other information about data types, preset values, restrictions or limitations.

- This improves the consistency of the analysis model and helps to reduce errors. The data dictionary defines information items unambiguously.
- The data dictionary content description tells us that such numbers are not part of the data that may be used.

CLASS BASED MODELING

Q.38. Explain class based modeling in details.

Ans. Class based modeling :

- Class-based modeling represents the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined.
- The elements of a class-based model include classes and objects, attributes, operations, class-responsibility-collaborator (CRC) models, collaboration diagrams, and packages.

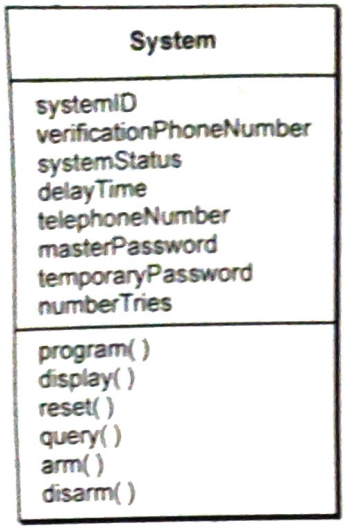


Fig. Class diagram for the system class

Identifying Analysis Classes :

- To identify classes by examining the usage scenarios developed as part of the requirements model and performing a "grammatical parse" on the use cases developed for the system to be built.
- Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
- Analysis classes details in one of the following ways :

(i) External entities (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.

(ii) Things (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.

(iii) Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.

(iv) Roles (e.g., manager, engineer, salesperson) played by people who interact with the system.

(v) Organizational units (e.g., division, group, team) that are relevant to an application.

(vi) Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.

(vii) Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

Specifying attributes :

- Attributes describe a class that has been selected for inclusion in the requirements model.
- In essence, it is the attributes that define the class that clarify what is meant by the class in the context of the problem space.
- To develop a meaningful set of attributes for an analysis class, you should study each use case and select those "things" that reasonably "belong" to the class.

Defining operations :

- Operations define the behavior of an object. Although many different types of operations exist, they can generally be divided into four broad categories:

(i) Operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting).

(ii) Operations that perform a computation.

(iii) Operations that inquire about the state of an object.

(iv) Operations that monitor an object for the occurrence of a controlling event.

Q.39. Explain the approach used for reviewing the class-responsibility-collaborator (CRC) model, by the representatives from the customer and software engineering organization.

CS : S-10(5M)

OR Explain CRC model index card with example. Also list the guidelines that can be applied for allocating responsibilities to classes.

CS : S-12(10M)

OR Explain class-responsibility-collaborator (CRC) modeling with example.

CT : S-12(7M), CS : S-13(6M), W-13(5M)

Ans. Class-responsibility-collaborator :

- Class-responsibility-collaborator (CRC) modelling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.
- A CRC model is really a collection of standard index cards that represent classes.
- The cards are divided into three sections. Along the top of the card we write the name of the class.
- In the body of the card we list the class responsibilities on the left and the collaborators on the right.
- The intent is to develop an organized representation of classes. Responsibilities are the attributes and operations that are relevant for the class.

- Collaborators are those classes that are required to provide a class with the information needed to complete a responsibility.
- In general, a collaboration implies either a request for information or a request for some action.

Class : Floor Plan	
Description	
Responsibility :	**Collaborator :**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors and windows	Wall
Shows position of video cameras	Camera

Fig. A CRC model index card

(1) **Classes :**

- It consider the categories Entity classes, also called model or business classes, are extracted directly from the statement of the problem (e.g., Floor Plan and Sensor).
- These classes typically represent things that are to be stored in a database and persist throughout the duration of the application.
- Boundary classes are used to create the interface that the user sees and interacts with as the software is used.
- For example, a boundary class called Camera Window would have the responsibility of displaying surveillance camera output for the Safe Home system.
- Controller classes manage a "unit of work" from start to finish. That is, controller classes can be designed to manage.

(2) **Responsibilities :**

- Basic guidelines for identifying responsibilities (attributes and operations).
- Wirfs-Brock and her colleagues suggest five guidelines for allocating responsibilities to classes:

(i) System intelligence should be distributed across classes to best address the needs of the problem.

(ii) Each responsibility should be stated as generally as possible.

(iii) Information and the behavior related to it should reside within the same class.

(iv) Information about one thing should be localized with a single class, not distributed across multiple classes.

(v) Responsibilities should be shared among related classes, when appropriate.

(3) **Collaborations :**

- Classes fulfill their responsibilities in one of two ways:

(i) A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility.

(ii) A class can collaborate with other classes.

- Wirfs-Brock define collaborations in the following way:

"Collaborations represent requests from a client to a server in fulfillment of a client responsibility. A collaboration is the embodiment of the contract between the client and the server."

- A single collaboration flows in one direction representing a request from the client to the server. From the client's point of view, each of its collaborations is associated with a particular responsibility implemented by the server.
- Collaborations are identified by determining whether a class can fulfill each responsibility itself.

BEHAVIORAL MODEL

Q.40. Explain behavioral model in details.

Ans. Behavioral Model :

- Behavioral models are models of the dynamic behavior of the system as it is executing.
- The behavioral model indicates how software will respond to external events or stimuli.
- To create the model, we should perform the following steps :

(i) Evaluate all use cases to fully understand the sequence of interaction within the system.

(ii) Identify events that drive the interaction sequence and understand how these events relate to specific objects.

(iii) Create a sequence for each use case.

(iv) Build a state diagram for the system.

(v) Review the behavioral model to verify accuracy and consistency.

- Identifying Events with the use case exchanged. A use case is examined for points of information exchange.

- In the above curves it provide the useful guidance when modularity is considered. We should modularize, but care should be taken to stay in the vicinity of M. Undermodularity or over modularity should be avoided.
- Five criteria that enable to evaluate a design method with respect to its ability to define an effective modular system:

(1) **Modular decomposability** : If a design method provides a systematic mechanism for decomposing the problem into sub problems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution.

(2) **Modular composability** : If a design method enables existing design components to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel.

(3) **Modular understandability** : If a module can be understood as a standalone unit (without reference to other modules), it will be easier to build and easier to change.

(4) **Modular continuity** : If small changes to the system requirements result in changes to individual modules, rather than systemwise changes, the impact of change-induced side effects will be minimized.

(5) **Modular protection** : If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized.

Q.51. What are the design heuristics for effective modularity?

CT: W-09(7M)

Ans. The program structure, for effective modularity, can be manipulated according to the following heuristics :

(i) Evaluate the first iteration of the program structure so that coupling will be reduced and cohesion will be improved. After developing the program structure, modules may be exploded or imploded for improving their independence. While an exploded module becomes two or more modules, an imploded module is formed by combining the processing implied by two or more modules, in the final program structure.

(ii) Attempt to minimize high fan-out structures; strive for fan-in with increase in depth. The structure shown inside the rectangle in Fig. does not use the factoring effectively. However, all modules are "pancaked" below a single control module. As a general case a more reasonable distribution of control is shown in the upper structure of Fig.

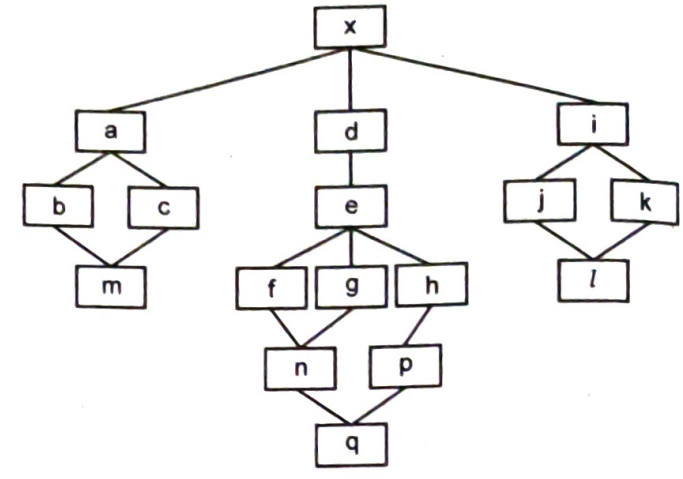


Fig. Program structures

(iii) Keep the scope of effect of a module within its scope of control. The "scope of effect" of module "e" can be defined as all other modules affected by module "e" decision, while the "scope of control" of module "e" is all modules that are subordinate (and ultimate subordinate) to module "e".

(iv) Evaluate module interfaces so as to reduce complexity and redundancy and to improve consistency.

(v) Define modules whose function is predicatable, at the same time, avoid modules that are overly restrictive. When a module can be treated as a black box, it is predictable, i.e., when the same external data will be produced with no effect of internal processing details.

(vi) Strive for "controlled entry" modules by avoiding "pathological connections". It warns against content coupling. Software is easily understandable and thus easily maintainable when interfaces of the module are constrained and controlled. (Pathological connection means branches or references into the middle of the module.)

Q.52. What is software architecture? Explain the model used to represent architectural design. **CS : W-09(7M)**

Ans. **Software Architecture :**

- The Software architecture is the structure or organization of program components , the manner in which these components interact, and the structure of data that are used by the components.

- In a broader sense, however, components can be generalized to represent major system elements and their interactions.
- The main purpose of software design is to derive an architectural rendering of a system. This rendering serves as a framework from which more detailed design activities are conducted. A set of architectural patterns enable a software engineer to reuse design level concepts.
- Following are the properties that should be specified as part of an architectural design :

(i) **Structural properties :** This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another.

(ii) **Extra-functional properties :** The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.

(iii) **Families of related systems :** The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.

- The architectural design can be represented using one or more of a number of different models. Structural models represent architecture as an organized collection of program components.
- Dynamic models address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.
- Process models focus on the design of the business or technical process that the system must accommodate.

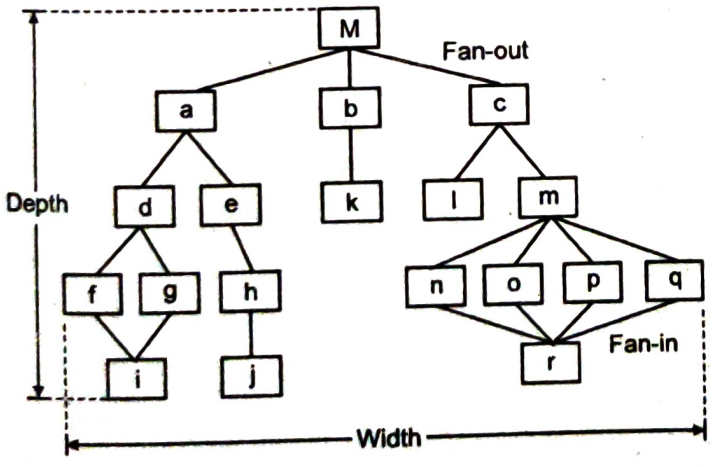


Fig. Structure terminology for a call and return architectural style

- At last, functional models can be used to represent the functional hierarchy of a system. A number of different architectural description languages (ADLs) have been developed to represent these models.
- Many different ADLs have been proposed, the majority provide mechanisms for describing system components and the manner in which they are connected to one another.

Q.53. What do you understand by the term functional independence? How to achieve functional independence in a software design?

CT: W-07(8M), CS: S-13(2M)

Ans. Functional Independence :

- Functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding.
- Functional independence is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.
- It represent in another way that, if we want to design software so that each module addresses a specific sub function of requirements and has a simple interface when viewed from other parts of the program structure.
- The importance of dependence is that the software with effective modularity, which is independent modules and it is easier to develop because function may be compartmentalized and interfaces are simplified.
- Independent modules is simple to maintain and for test because secondary effects caused by design or code modification are limited, error propagation is reduced and reusable modules are possible.
- To summarize, functional independence is a key to good design, and design is the key to software quality.
- Independence is measured using two qualitative criteria: cohesion and coupling. Cohesion is a measure of the relative functional strength of a module. Coupling is a measure of the relative interdependence among modules.

Q.54. Explain the concept of cohesion and coupling. How can an ideal design in which modules have high cohesion and low coupling be achieved ? *CT : S-06,08(6M), W-12(9M), CS : S-09(13M)*

OR Enumerate and explain the different types of coupling that a module may exhibit. *CT: W-07(6M)*

OR How good design objective can be achieved in software engineering using coupling and cohesion? *CS: S-10(10M)*

OR Write short note on cohesion and coupling. *CT : S-11(3M), W-11(6M)*

OR Why the low coupling is required for ideal design? Justify with coupling spectrum. *CT: S-13(7M)*

OR Explain the terms cohesion and coupling in detail. *CS : W-13(2M), W-09(9M), W-12(3M), CT: S-10(13M)*

Ans. Cohesion:

- It is a natural extension of the information hiding concept. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.
- The scale for cohesion is nonlinear. That is, low-end cohesiveness is much "worse" than middle range, which is nearly as "good" as high-end cohesion.
- A designer need not be concerned with categorizing cohesion in a specific module. Rather, the overall concept should be understood and low levels of cohesion should be avoided when modules are designed.
- A module that performs tasks that are related logically (e.g., a module that produces all output regardless of type) is logically cohesive.
- When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits temporal cohesion.
- As an example of low cohesion, consider a module that performs error processing for an engineering analysis package. The module is called when computed data exceed pre specified bounds.
- It performs the following tasks:

(i) Computes supplementary data based on original computed data.

(ii) Produces an error report (with graphical content) on the user's workstation.

(iii) Performs follow-up calculations requested by the user.

(iv) Updates a database.

(v) Enables menu selection for subsequent processing.

- The preceding tasks are loosely related, each is an independent functional entity that might best be performed as a separate module.
- Combining the functions into a single module can serve only to increase the likelihood of error propagation when a modification is made to one of its processing tasks.
- Moderate levels of cohesion are relatively close to one another in the degree of module independence. When processing elements of a module are related and must be executed in a specific order, procedural cohesion exists.
- When all processing elements concentrate on one area of a data structure, communicational cohesion is present.

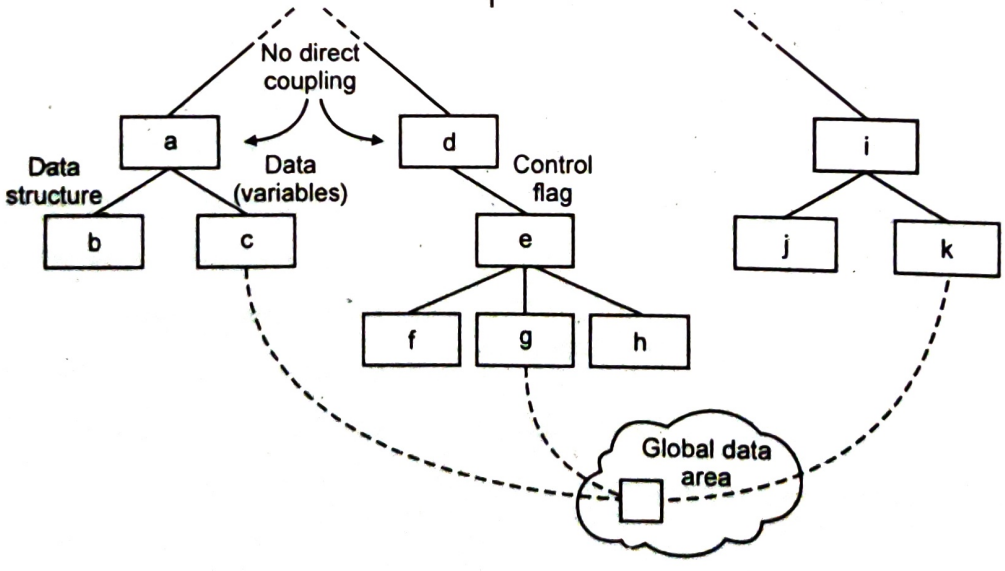


Fig. Types of coupling

Coupling :

- Coupling is a measure of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.
- In software design, we strive for lowest possible coupling. Simple connectivity among modules results in software that is easier to understand and less prone to a "ripple effect", caused when errors occur at one location and propagate through a system.
- The above figure provides examples of different types of module coupling. Modules a and d are subordinate to different modules. Each is unrelated and therefore no direct coupling occurs. Module c is subordinate to module a and is accessed via a conventional argument list, through which data are passed.
- A variation of data coupling, called stamp coupling, is found when a portion of a data structure (rather than simple arguments) is passed via a module interface. This occurs between modules b and a. At moderate levels, coupling is characterized by passage of control between modules.
- Control coupling is very common in most software designs and is shown in figure, where a "control flag" is passed between modules d and e.
- The highest degree of coupling, content coupling, occurs when one module makes use of data or control information maintained within the boundary of another module. Secondarily, content coupling occurs when branches are made into the middle of a module. This mode of coupling can and should be avoided.
- The coupling modes just discussed occur because of design decisions made when structure was developed. Variants of external coupling, however, may be introduced during coding.

Q.55. Explain cohesion spectrum in detail. *CT: S-12(7M)*

Ans. Cohesion spectrum :

- Cohesion may be represented as a "spectrum."
- The scale for cohesion is nonlinear. That is, low-end cohesiveness is much "worse" than middle range, which is nearly as "good" as high-end cohesion.
- In practice, a designer need not be concerned with categorizing cohesion in a specific module.
- At the low (undesirable) end of the spectrum, we encounter a module that performs a set of tasks that relate to each other loosely, if at all. Such modules are termed coincidentally cohesive.
- A module that performs tasks that are related logically (e.g. a module that produces all output regardless of type) is logically cohesive.
- When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits temporal cohesion.
- As an example of low cohesion, consider a module that performs error processing for an engineering analysis package. The module is called when computed data exceed prespecified bounds. It performs the following tasks:

(i) Computes supplementary data based on original computed data.

(ii) Produces an error report (with graphical content) on the user's workstation.

(iii) Performs follow-up calculations requested by the user.

(iv) Updates a database.

(v) Enables menu selection for subsequent processing.

- Moderate levels of cohesion are relatively close to one another in the degree of module independence.
- When processing elements of a module are related and must be executed in a specific order, procedural cohesion exists.
- When all processing elements concentrate on one area of a data structure, communicational cohesion is present.
- High cohesion is characterized by a module that performs one distinct procedural task.

DESIGN MODEL

Q.56. How do you obtain the optimum number of modules during design modeling? Support your answer with a neat diagram.

CS: S-10(8M)

Ans. Design model :

- Design or system models, show the objects or object classes in a system. They also show the associations and relationships between these entities.

SOLVED QUESTION BANK

[Sequence given as per syllabus]

INTRODUCTION

Software testing is the process of executing a program or system with the intent of finding errors. It involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Software testing depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted. Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications. The data collected through testing can also provide an indication of the software's reliability and quality.

SOFTWARE TESTING : TESTING FUNDAMENTALS

Q.1. Explain the software testing fundamental.

Ans. Software testing fundamentals are as follows :

(1) Testability : "Software testability is simply how easily it can be tested."

(2) Operability : "The better it works, the more efficiently it can be tested." If a system is designed and implemented with quality in mind, relatively few bugs will block the execution of tests, allowing testing to progress without fits and starts.

(3) Observability : "What you see is what you test." Inputs provided as part of testing produce distinct outputs. System states and variables are visible or queriable during execution. Incorrect output is easily identified. Internal errors are automatically detected and reported. Source code is accessible.

(4) Controllability : "The better we can control the software, the more the testing can be automated and optimized." All possible outputs can be generated through some combination of input, and I/O formats are consistent and structured. All code is executable through some combination of input.

(5) Decomposability : "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting." The software system is built from independent modules that can be tested independently.

(6) Simplicity : "The less there is to test, the more quickly we can test it." The program should exhibit functional simplicity (e.g., the feature set is the minimum necessary to meet requirements) structural simplicity and code simplicity.

(7) Stability : "The fewer the changes, the fewer the disruptions to testing." Changes to the software are infrequent, controlled when they do occur, and do not invalidate existing tests. The software recovers well from failures.

(8) Understandability : "The more information we have, the smarter we will test." The architectural design and the dependencies between internal, external, and shared components are well understood.

Q.2. Explain the characteristics of testing.

OR What do you mean by testing of project ? ***CT : S-07(2M)***

OR What are the attributes of a good test ? Explain. ***CS : S-10(2M)***

OR Explain software testing in details.

OR What is testing ? ***CS : W-10(3M)***

Ans. Software testing :

- Testing is the process to detect the defects and minimize the risk associated with the residual defects.
- Software testing have a probability of finding an error and minimize the risk associated with that error.
- To achieve this goal, the tester must understand the software and attempt to develop a mental picture of how the software might fail. Ideally, the classes of failure are probed.
- For example, one class of potential failure in a graphical user interface is the failure to recognize proper mouse position.
- A set of tests would be designed to exercise the mouse in an attempt to demonstrate an error in mouse position recognition.
- The attributes of a good test are as follows :

(i) A good test is not redundant. Testing time and resources are limited. There is no point in conducting a test that has the same purpose as another test. Every test should have a different purpose.

(ii) A good test should be "best of breed". In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests. In such cases, the test that has the highest likelihood of uncovering a whole class of errors should be used.

(iii) A good test should be neither too simple nor too complex. Although it is sometimes possible to combine a series of tests into one test case, the possible side effects associated with this approach may mask errors. In general, each test should be executed separately.

Q.3. Explain various testing principles. **CS :W-12(5M)**

OR. What are the guidelines those leads to a successful software testing? **CS : S-11(5M)**

Ans. The testing principles are as follows :

(1) All tests should be traceable to customer requirements :

The objective of software testing is to uncover errors. It follows that the most severe defects from the customer's point of view are those that cause the program to fail to meet its specification.

(2) Tests should be planned long before testing begins :

Test planning can begin as soon as the requirements model is complete. Detailed definition of test cases can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.

(3) The Pareto principle applies to software testing :

The Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.

(4) Testing should begin "in the small" and progress toward testing "in the large " :

The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

(5) Exhaustive testing is not possible :

The number of path permutations for even a moderately sized program is exceptionally large for this reason, it is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised.

(6) To be most effective, testing should be conducted by an independent third party :

By most effective, we mean testing that has the highest probability of finding errors the primary objective of testing. The software engineer who created the system is not the best person to conduct all tests for the software.

Q.4. Write a note on testing strategy.

OR What are different strategies and levels of software testing ? **CT : W-07, S-09(1M)**

OR Explain in context of a spiral, the software testing strategy for conventional software architecture. **CS : S-10(5M), S-12(6M), S-13(7M)**

Ans. Testing strategy :

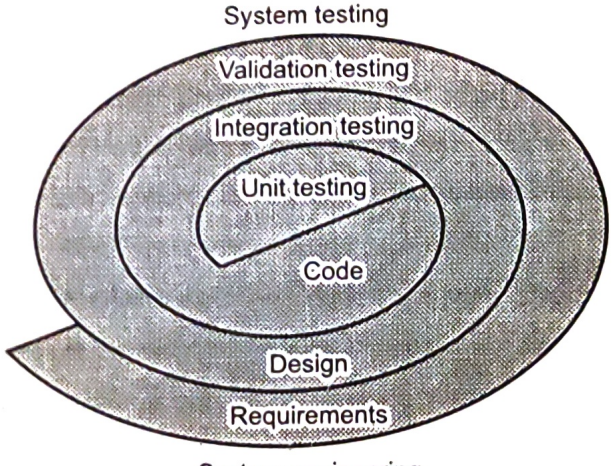


Fig. (a) Testing Strategy

- The activities that can be planned in advance and conducted systematically are known as testing strategy. Therefore software testing is a set of steps into which we can place specific test case design techniques and testing methods should be defined for the software process.
- Software testing strategies have been proposed that all provide the software developer with a template for testing and all have the following generic characteristics:

(i) Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.

(ii) Different testing techniques are appropriate at different points in time.

(iii) Testing is conducted by the developer of the software and an independent test group.

(iv) Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

(v) A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

(vi) A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems must surface as early as possible.

Pressman :

- A strategy for software testing may also be viewed in the context of the spiral.
- Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code.
- Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture.
- Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed.
- Finally, we arrive at system testing, where the software and other system element are tested as a whole. To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.
- Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of four steps that are implemented sequentially. The steps are shown in Fig. (b).
- Initially, tests focus on each component individually, ensuring that it functions properly as a unit. Hence, the name unit testing. Unit testing make heavy use of white-box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection.
- Next, components must be assembled or integrated to form the complete software package. Integration testing addresses the issues associated with the dual problems of verification and program construction. Black-box test case design techniques are the most prevalent during integration, although a limited amount of white-box testing may be used to ensure coverage of major control paths.
- After the software has been integrated (constructed), a set of high-order tests are conducted. Validation criteria (established during requirements analysis) must be tested. Validation testing provides final assurance that software meets all functional, behavioural, and performance requirements. Black-box testing techniques are used exclusively during validation.

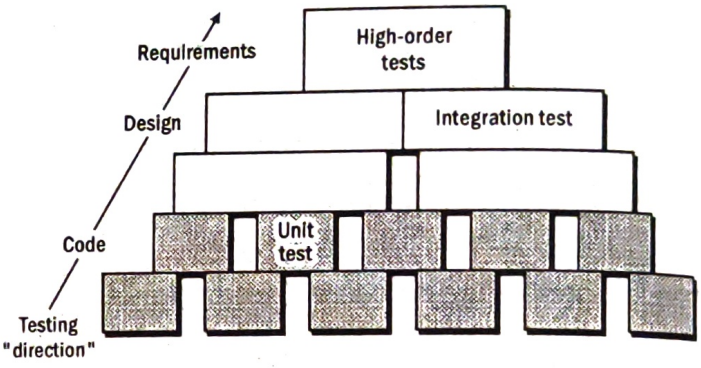


Fig.(b) Software testing steps

- The last high-order testing step falls outside the boundary of software engineering and into the broader context of computer system engineering.
- Software, once validated, must be combined with other system elements (e.g. hardware, people, databases).
- System testing verifies that all elements mesh properly and that overall system function / performance is achieved.

BLACK-BOX TESTING

Q.5. Explain black box testing and its method.

CT : W-06(7M), S-08, W-08(4M), W-09(5M), W-11(2M), S-07,11(3M)
CS : S-09,11(4M), W-09(7M), W-10(5M), W-11(6M)

OR Define black box testing. *CS : W-12(1M)*

OR Explain in detail about boundary value analysis.

CS : W-12(3M), CT : S-14(3M)

OR Write a short note on orthogonal array testing.

CS : W-12(2M)

OR **Explain equivalence partitioning and boundary value analysis techniques for black box testing.** **CS : S-13,W-13(6M)**

Ans. **Black-box testing :**

- Black-box testing is also called closed box, opaque box, and behavioral testing. It focuses on the functional requirements of the software. That is, black-box testing techniques enable us to derive sets of input conditions that will fully exercise all functional requirements for a program.
- Black-box testing attempts to find errors in the following categories:

(i) Incorrect or missing functions.

(ii) Interface errors.

(iii) Errors in data structures or external database access.

(iv) Behavior or performance errors.

(v) Initialization and termination errors.

- By applying black-box techniques, we derive a set of test cases that satisfy the following criteria :

(i) Test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieve reasonable testing.

(ii) Test cases that tell you something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

- The various black-box testing methods are as follows :

(1) Graph-Based Testing Methods :

- First step in black-box testing is to understand the objects that are modeled in software and the relationships that connect these objects. Once this has been accomplished, the next step is to define a series of tests that verify "all objects have the expected relationship to one another".
- Software testing begins by creating a graph of important objects and their relationships and then devising a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.

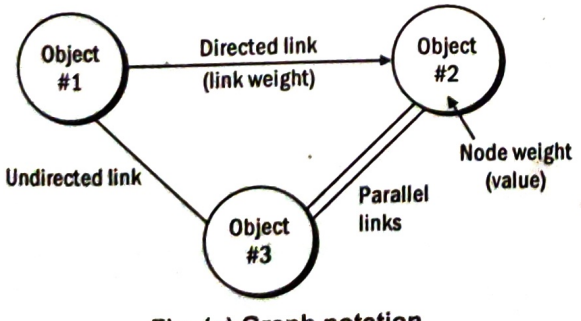


Fig. (a) Graph notation

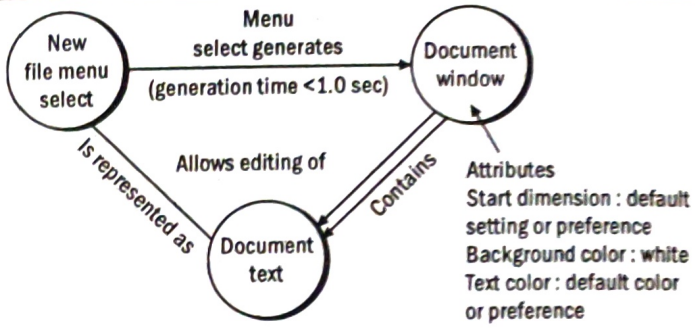


Fig. (b) Simple example

- The symbolic representation of a graph is shown in figure. Nodes are represented as circles connected by links that take a number of different forms.
- A directed link (represented by an arrow) indicates that a relationship moves in only one direction. A bidirectional link, also called a symmetric link.
- Symmetric link implies that the relationship applies in both directions. Parallel links are used when a number of different relationships are established between graph nodes.

(2) Equivalence Partitioning :

- Equivalence partitioning is one of the black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.
- An ideal test case single-handedly uncovers a class of errors that might otherwise require many test cases to be executed before the general error is observed.
- In equivalence partitioning the test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition. If a set of objects can be linked by relationships that are symmetric, transitive, and reflexive, an equivalence class is present.
- An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition.
- Equivalence classes may be defined according to the following guidelines:

(i) If an input condition specifies a range, one valid and two invalid equivalence classes are defined.

(ii) If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.

(iii) If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.

(iv) If an input condition is Boolean, one valid and one invalid class are defined. By applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed. Test cases are selected so that the largest number of attributes of an equivalence class is exercised at once.

(3) Boundary Value Analysis :

- In boundary value analysis a greater number of errors occurs at the boundaries of the input domain rather than in the "center." It is for this reason that boundary value analysis (BVA) has been developed as a testing technique.
- Boundary value analysis leads to a selection of test cases that exercise bounding values.
- Boundary value analysis is a test-case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class.
- Guidelines for BVA are similar in many respects to those provided for equivalence partitioning which are as follows :

(i) If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and just below a and b.

(ii) If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

- Apply guidelines 1 and 2 to output conditions. For example, assume that a temperature versus pressure table is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.
- If internal program data structures have prescribed boundaries be certain to design a test case to exercise the data structure at its boundary.

(4) Comparison Testing :

- When the reliability of software is absolutely critical, in such applications redundant hardware and software are often used to minimize the possibility of error. When redundant software is developed, separate software engineering teams develop independent versions of an application using the same specification.
- Each version can be tested with the same test data to ensure that all provide identical output. Then all versions are executed in parallel with real-time comparison of results to ensure consistency.
- Independent versions of software be developed for critical applications, even when only a single version will be used in the delivered computer-based system.
- Comparison testing is not fullproof. If the specification from which all versions have been developed is in error, all versions will likely reflect the error. In addition, if each of the independent versions produces identical but incorrect results, condition testing will fail to detect the error.

(5) Orthogonal Array Testing :

- Whenever there is having many applications in which the input domain is relatively limited. That is, the number of input parameters is small and the values that each of the parameters may take are clearly bounded.
- When these numbers are very small (e.g., three input parameters taking on three discrete values each), it is possible to consider every input permutation and exhaustively test the input domain.
- However, as the number of input values grows and the number of discrete values for each data item increases, exhaustive testing becomes impractical or impossible.
- Orthogonal array testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.
- The orthogonal array testing method is particularly useful in finding region faults an error category associated with faulty logic within a software component.

Q.6. What is SRS? List out the advantages of SRS standards. Why SRS is known as the black box specification of a system?

Ans. SRS :

- A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development.
- The SRS fully describes what the software will do and how it will be expected to perform.
- An SRS is basically an organization's understanding of a customer or potential client's system requirements and dependencies at a particular point in time prior to any actual design or development work.

- It's important to note that an SRS contains functional and nonfunctional requirements only.

Advantages of SRS standards :

(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) **Provide a basis for developing the software design :**

The SRS is the most important document of reference in developing a design.

(3) **Reduce the development effort :**

- The preparation of the SRS forces the various concerned groups in the customer's organisation to thoroughly consider all of the requirements before design work begins. A complete and correct SRS reduces effort waisted on 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

(4) **Provide a basis for estimating costs and schedules :**

The description 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

(5) **Provide a baseline for validation and verification :**

- Organisations can develop their test documentation 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

(6) **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 organisation and suppliers find it easier to transfer it to new customers

(7) **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 product evaluation.
- Software requirement specification is also known as 'black-box' testing techniques because they view the software as a black-box with inputs and outputs.

Q.7. Define each of the following terms :

(1) Adaptive maintenance.

(2) Error.

Ans.

(1) Adaptive maintenance :

- Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system.
- Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system.
- The term environment in this context refers to the conditions and the influences which act on the system. For example, business rules, work patterns, and government policies have a significant impact on the software system.

(2) Error :

- One common definition of a software error is a mismatch between the program and its specification.
- In other words, we can say, a software error is present in a program when the program does not do what its end user expects, Error refers to the difference between the actual output of software and the correct output.

WHITE BOX TESTING

Q.8. Explain white box testing and its methods.

CT : S-06(7M), W-06, S-10 (6M), S-07(2M), S-08(9M), W-08(3M), W-09,11(5M), S-11, W-12(4M)

CS : W-08(8M), S-09, W-10(5M), W-09,11(6M)

OR Explain with example how cyclomatic complexity is calculated. *CS : S-14(7M)*

OR Define white-box testing. *CS : W-12(1M)*

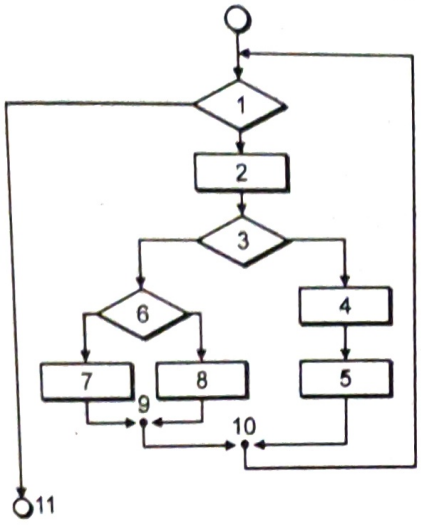
OR What is basis path testing? *CS : S-12(5M)*

OR What is Mccabe's cyclomatic complexity? How it is useful for white box testing? *CT : S-14(6M)*

OR How cyclomatic complexity useful in basis path testing? Explain with example. *CS : S-10(7M), S-11(6M)*

OR What is cyclomatic complexity? Calculate cyclomatic complexity for following flowchart by all three methods.

CS : W-12(8M), S-13(7M)



Ans. **White box testing :**

- White-box testing is sometimes called clear box testing, glass-box testing and transparent testing. It is a test case design method that uses the control structure of the procedural design to derive test cases.
- Using white-box testing methods, the software engineer can derive test cases that :

(i) Guarantee that all independent paths within a module have been exercised at least once.

(ii) Exercise all logical decisions on their true and false sides.

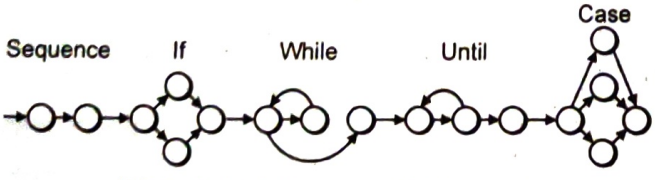
(iii) Execute all loops at their boundaries and within their operational bounds.

(iv) Exercise internal data structures to ensure their validity.

Basis Path Testing /Techniques of white box testing are as follows :

(1) Flow Graph Notation :

- In software component, before the basis path method can be introduced, a simple notation for the representation of control flow, called a flow graph (or program graph) must be introduced.



Where each circle represents one or more nonbranching PDL or source code statements

Fig. (a) Flow graph notation

- A sequence of process boxes and a decision diamond can map into a single node. The arrows on the flow graph, called edges or links, represent flow of control and are analogous to flowchart arrows.
- An edge must terminate at a node, even if the node does not represent any procedural statements.
- Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph as a region.

(2) Cyclomatic Complexity :

- Cyclomatic complexity is one of the techniques of white box testing. Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.
- When used in the context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

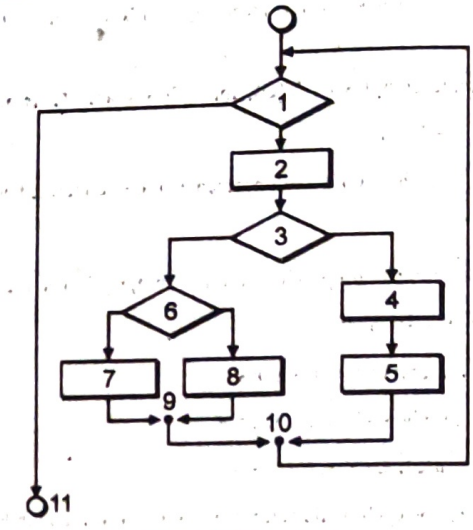


Fig. (b) Flowchart

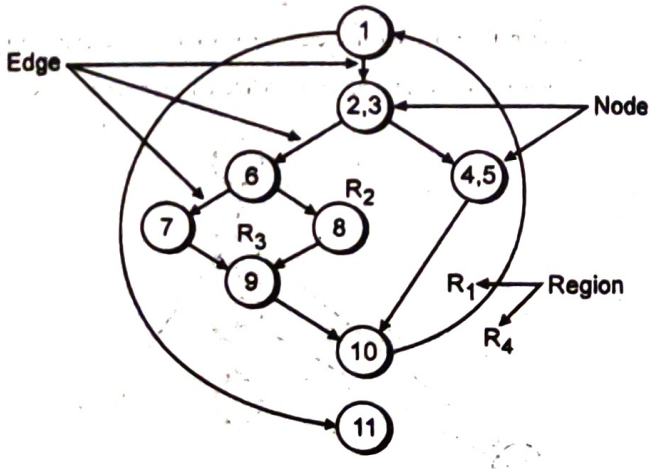


Fig. (c) Flow graph

- When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined. For example, a set of independent paths for the flow graph.

(i) path 1: 1-11

(ii) path 2: 1-2-3-4-5-10-1-11

(iii) path 3: 1-2-3-6-8-9-10-1-11

(iv) path 4: 1-2-3-6-7-9-10-1-11

- Note that each new path introduces a new edge.

The path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 is not considered to be an independent path because it is simply a combination of already specified paths and does not traverse any new edges.

- Cyclomatic complexity has a foundation in graph theory and provides us with an extremely useful software metric. Complexity is computed in one of three ways :

(i) The number of regions of the flow graph correspond to the cyclomatic complexity.

(ii) Cyclomatic complexity, V(G), for a flow graph, G, is defined as V(G) = E – N + 2.

where E is the number of flow graph edges, N is the number of flow graph nodes.

(iii) Cyclomatic complexity, V(G), for a flow graph, G, is also defined as: V(G) = P + 1

where, P is the number of predicate nodes contained in the flow graph G.

- The cyclomatic complexity can be computed using each of the algorithms just noted:

(i) The flow graph has four regions.

(ii) V(G) = 11 edges - 9 nodes + 2 = 4.

(iii) V(G) = 3 predicate nodes + 1 = 4.

Therefore, the cyclomatic complexity of the flow graph in Fig.(c) is 4.

(3) Deriving Test Cases :

- The basis path testing method can be applied to a procedural design or to source code, we present basis path testing as a series of steps.

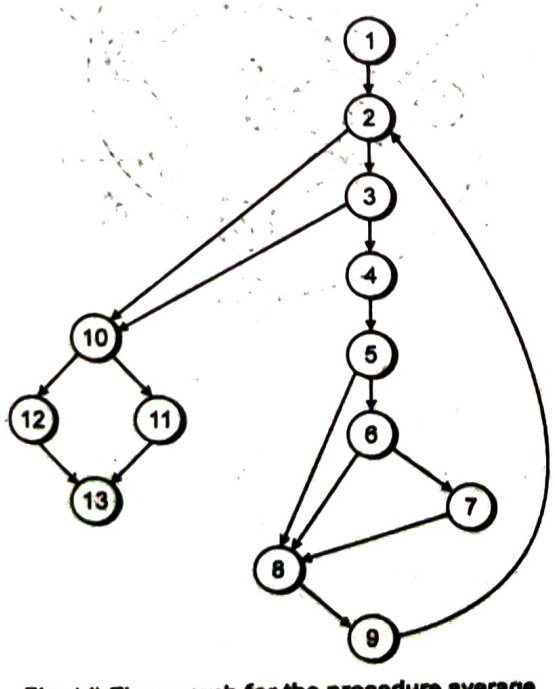


Fig. (d) Flow graph for the procedure average

- The following steps can be applied to derive the basis set :

(a) Using the design or code as a foundation, draw a corresponding flow graph :

A flow graph is created by numbering those PDL statements that will be mapped into corresponding flow graph nodes.

(b) Determine the cyclomatic complexity of the resultant flow graph :

It should be noted that V(G) can be determined without developing a flow graph by counting all conditional statements in the PDL (for the procedure average, compound conditions count as two) and adding 1.

V(G) = 6 regions

V(G) = 17 edges – 13 nodes + 2 = 6

V(G) = 5 predicate nodes + 1 = 6

(c) Determine a basis set of linearly independent paths :

- The value of V(G) provides the number of linearly independent paths through the program control structure. In the case of procedure average, we expect to specify six paths:

Path 1: 1-2-10-11-13

Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

Path 4: 1-2-3-4-5-8-9-2-. . .

Path 5: 1-2-3-4-5-6-8-9-2-. . .

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

- The ellipsis following paths 4, 5, and 6 indicates that any path through the remainder of the control structure is acceptable. It is often worthwhile to identify predicate nodes as an aid in the derivation of test cases. In this case, nodes 2, 3, 5, 6, and 10 are predicate nodes.

(d) Prepare test cases that will force execution of each path in the basis set :

Data should be chosen so that conditions at the predicate nodes are appropriately set as each path is tested.

(4) Graph Matrices :

- To develop a software tool that assists in basis path testing, a data structure, called a graph matrix is used.
- A graph matrix is a square matrix whose size is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node, and matrix entries correspond to connections between nodes. A simple example of a flow graph and its corresponding graph matrix.

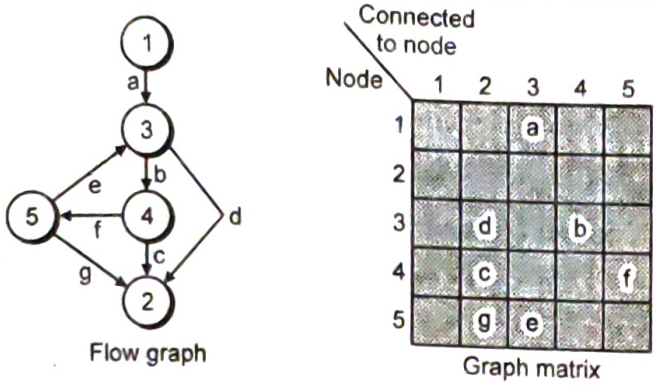


Fig. (e) Graph matrix

- Each node on the flow graph is identified by numbers, while each edge is identified by letters. A letter entry is made in the matrix to correspond to a connection between two nodes. For example, node 3 is connected to node 4 by edge b.
- To this point, the graph matrix is nothing more than a tabular representation of a flow graph. However, by adding a link weight to each matrix entry, the graph matrix can become a powerful tool for evaluating program control structure during testing.
- The link weight provides additional information about control flow. In its simplest form, the link weight is 1.

Q.9. Differentiate between black box testing and white box testing.

CS : S-10(5M)

Ans.

Sr. No.	Black box Testing	White box testing
(1)	Black box testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester.	White box testing is a software testing method in which the internal structure, design, implementation of the item being tested is known to the tester.
(2)	It is also called as functional testing and behaviour testing.	It is also known as glass box or clear box testing or transparent testing.
(3)	It performs test at software interface.	It performs close examination of procedural design.
(4)	It is done by independent software testers.	It is done by software developers.
(5)	It performs on the basis of software requirement specification.	It performs using details design.

Q.10. For a given PDL (Program Design Language) calculate the cyclomatic complexity :

PDL

Procedure: sort

1: do while record remain

Read record;

2: if record field 1=0

3: then process record;

store in buffer;

increment counter;

4: else if record field 2=0

5: then reset counter;

6: else process record;

Store in file;

7a: end if

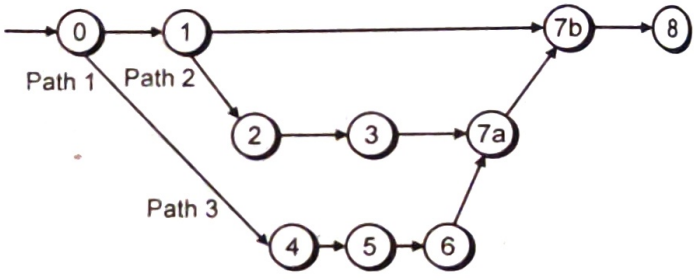
end if

7b: end do

8: end

CT : S-13(7M)

Ans.



The cyclomatic complexity

V(G) = E – N + 2

= 11 – 10+2

= 3 (No. of Path)

Q.11.(A) Apply the basis path technique to a video sales and rental shop example. Consider the following procedure for reserving a copy of a movie video.

Draw the flow graph and calculate cyclomatic complexity.

Procedure : video copy (result)

If (status = "available") OR

((status = "rented") AND

(return Date ≤ request Date))then

Ans. Integration testing :

- Integration testing is the testing process in software testing to verify that when two or more modules interact and produced result satisfies with its original functional requirement or not. Integrated testing comes under in black box testing.
- Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing.
- Integrated testing will start after completion of unit testing. Software testing engineer is performing integration testing.
- Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct interfaces are more likely to be tested completely and a systematic test approach may be applied.
- A number of different incremental integration strategies are as follows:

(1) Top-down integration :

- Top-down integration testing is an incremental approach to construction of program structure.
- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).
- Top-down integration testing is an integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated.
- Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.
- The replacement for the 'called' modules is known as 'Stubs' and is also used when the software needs to interact with an external system.

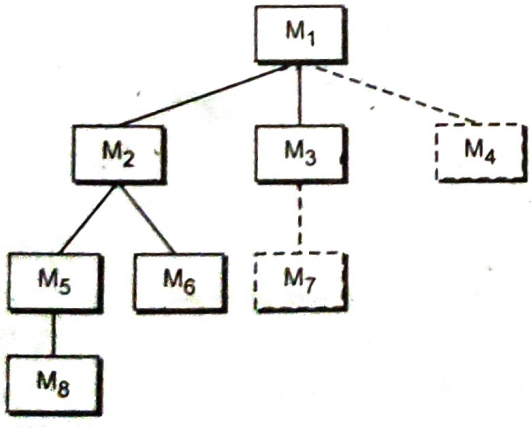


Fig. (a) Top-down integration

- Depth first integration would integrate all components on a major control path of the structure. Selection of a major path is somewhat arbitrary and depends on application-specific characteristics.
- For example, selecting the left hand path, components M1, M2 , M5 would be integrated first. Next, M8 or M6 would be integrated.
- Breadth-first integration incorporates all components directly subordinate at each level, moving across the structure horizontally.
- From the Fig.(a) components M2, M3, and M4 (a replacement for stub S4) would be integrated first. The next control level, M5, M6, and so on.
- The integration process is performed in a series of six steps :

(i) The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.

(ii) Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.

(iii) Tests are conducted as each component is integrated.

(iv) On completion of each set of tests, another stub is replaced with the real component.

(v) Regression testing may be conducted to ensure that new errors have not been introduced.

(vi) If depth-first integration is selected, a complete function of the software may be implemented.

- The major disadvantage of top-down approaches is the need for stubs and the difficulties that are linked with them. Problems linked with stubs maybe offset by the advantage of testing major control functions early.

(2) Bottom-up integration :

- In bottom up integration testing, module at the lowest level are developed first and other modules which go towards the 'main' program are integrated and tested one at a time.
- Bottom up integration also uses test drivers to drive and pass appropriate data to the lower level modules. As and when code for other module gets ready, these drivers are replaced with the actual module.
- In this approach, lower level modules are tested extensively thus make sure that highest used module is tested properly.

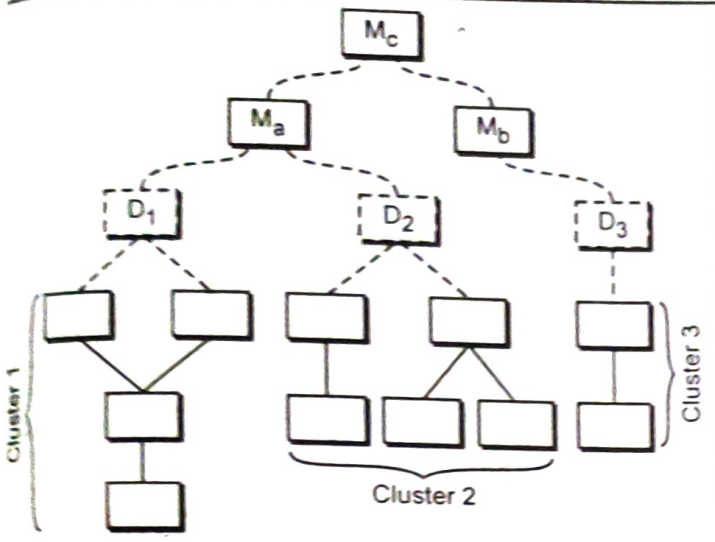


Fig. (b) Bottom-up integration

- A bottom-up integration strategy may be implemented with the following steps :

(i) Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub function.

(ii) A driver (a control program for testing) is written to coordinate test case input and output.

(iii) The cluster is tested.

(iv) Drivers are removed and clusters are combined moving upward in the program structure, components are combined to form clusters 1, 2, and 3. Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to M_a drivers.

(v) D1 and D2 are removed and the clusters are interfaced directly to M_a. Similarly, driver D3 for cluster 3 is removed prior to integration with module M_b .Both M_a and M_b will ultimately be integrated with component M_c and so on.

- The major drawback of bottom-up integration is that the program does not exist until the last module is included.

Q.21. Explain regression testing. *CT : W-07, S-09(3M), S-14(4M)*

Ans. Regression testing :

- Each time a new module is added as part of integration testing, software changes. New data flow paths are establishment changes may cause problems with functions that previously worked flawlessly.
- Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture tools.
- A test suite is developed incrementally as a program is developed, we can always run regression tests to check that changes to the program have not introduced new bugs.
- The regression test suite contains three different classes of test cases as follows :

(i) A representative sample of tests that will exercise all software functions.

(ii) Additional tests that focus on software functions that are likely to be affected by the change.

(iii) Tests that focus on the software components that have been changed.

Q.22. Explain smoke testing.

Ans. Smoke testing :

- Smoke testing is an integration testing approach that is commonly used when "shrinkwrapped" and software products are being developed.
- Smoke test helps in exposing integration and major problems early in the cycle. It can be conducted on both newly created software and enhanced software.
- Smoke test is performed manually or with the help of automation tools/scripts. If builds are prepared frequently, it is best to automate smoke testing.
- Smoke testing covers most of the major functions of the software but none of them in depth. The result of this test is used to decide whether to proceed with further testing. If the smoke test passes, go ahead with further testing.
- The smoke testing approach encompasses the following activities:

(i) Software components that have been translated into code are integrated into a "build." A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.

(ii) A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule.

(iii) The build is integrated with other builds and the entire product is smoke tested daily. The integration approach may be top down or bottom up.

Advantages :

(i) It helps to find issues introduced in integration of modules.

(ii) It helps to find issues in the early phase of testing.

(iii) It helps to get confidence to tester that fixes in the previous builds not breaking major features.

VALIDATION TESTING

Q.23. Write a short note on validation testing.

CT : W-07, S-09,11,(3M), W-11, W-13(2M)

CS : S-11(2M), W-11(3M), W-12(6M), W-13(5M)

Ans. Validation testing :

- Software functions in a manner that can be reasonably expected by the customer.
- Software validation is the process of testing a software to check whether it satisfies the customer needs or not. This testing is done during or at the end of the process of software development.
- Validation is basically done by the testers during the testing. While validating the product if some deviation is found in the actual result from the expected result then a bug is reported or an incident is raised.
- Validation is done at the end of the development process and takes place after verifications are completed.
- Determination of correctness of the final software product by a development project with respect to the user needs and requirements.
- Software validation is achieved through a series of black-box tests that demonstrate confirmity with requirements.
- For each validation test case has been conducted, one of two possible conditions exist :

(i) The function or performance characteristics confirm to specification and are accepted.

(ii) A deviation from specification is uncovered and a deficiency list is created. Deviation or error discovered at this stage in a project can rarely be corrected prior to scheduled delivery.

- It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

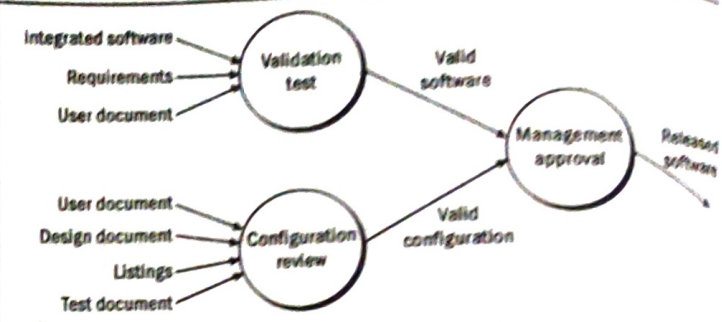


Fig. Audit/configuration review

- An important element of the validation process is a configuration review, also called as 'Audit'.
- The intent of the review, is to ensure that all the elements of the software configuration have been properly developed, cataloged and have the necessary detail to support the maintenance phase of the software life cycle.

Q.24. Write a short note on software verification and integration.

CT : S-11, W-13(2M), W-11(1M)

CS : S-11(2M)

Ans. Software verification :

- Software validation ensures that the product actually meets the user's needs, and that the specifications were correct in the first place, while software verification is ensuring that the product has been built according to the requirements and design specifications. Software validation ensures that "you built the right thing".
- Software verification ensures that "you built it right". Software validation confirms that the product, as provided, will fulfill its intended use.
- Validation checks that the product design satisfies or fits the intended use, i.e., the software meet the user requirements. This is done through dynamic testing and other forms of review.

Software integration :

- Software integration is defined as the process of bringing together the component subsystems into one system and ensuring that the subsystems function together as a system.
- In information technology, systems integration is the process of linking together different computing systems and software applications physically or functionally, to act as a coordinated whole.
- A system is an aggregation of subsystems cooperating so that the system is able to deliver the overarching functionality. System integration involves integrating existing often disparate systems.

- Software integration (SI) is also about adding value to the system, capabilities that are possible because of interactions between subsystems.

Q.25. Explain Alpha and Beta testing.

CS : S-11(3M), W-11(5M), S-13(2M)

Ans. **Alpha testing :**

- The alpha test is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems.
- Alpha tests are conducted in a controlled environment.
- Alpha testing is definitely performed and carried out at the developing organizations location with the involvement of developers.
- Alpha testing is always performed at the time of Acceptance testing when developers test the product and project to check whether it meets the user requirements or not.

Beta testing :

- The beta test is conducted at one or more customer sites by the end user of the software. Unlike alpha testing, the developer is generally not present.
- Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems that are encountered during beta testing and reports these to the developer at regular intervals.
- As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.

Q.26. Write the difference between Alpha and Beta testing.

CT : W-08(6M)

Ans.

Sr. No.	Alpha Testing	Beta Testing
(1)	It is always performed by the developers at the software development site.	It is always performed by the customers at their own site.
(2)	Sometimes it is also performed by independent testing team.	It is not performed by independent testing team.
(3)	Alpha testing is not open to the market and public	Beta testing is always open to the market and public.
(4)	It is conducted for the software application and project.	It is usually conducted for software product.
(5)	It is always performed in virtual environment.	It is performed in real time environment.
(6)	It is always performed within the organization.	It is always performed outside the organization.
(7)	It is the form of Acceptance Testing.	It is also the form of Acceptance Testing.
(8)	Alpha testing is definitely performed and carried out at the developing organizations location with the involvement of developers.	Beta testing (field testing) is performed and carried out by users or we can say people at their own locations and site using customer data.
(9)	It comes under the category of both White Box Testing and Black Box Testing.	It is only a kind of Black Box Testing.
(10)	Alpha testing is always performed at the time of Acceptance Testing when developers test the product and project to check whether it meets the user requirements or not.	Beta testing is always performed at the time when software product and project are marketed.

SYSTEM TESTING

Q.27. Write a note on the following :

(a) System Testing. *CT : W-07, S-09(3M)*

(b) Recovery Testing.

(c) Security Testing.

(d) Stress Testing.

(e) Performance Testing.

(f) Functional Testing.

OR **List and explain in brief system testing techniques.**

CS : W-13(4M)

OR **Explain the four types of system testing performed to fully exercise the computer based system.** *CS : S-10(8M)*

OR **Explain system testing and different methods of it.**

Ans.

(a) System Testing :

- System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems.
- It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties.
- For large systems, this may be a multi-stage process where components are integrated to form subsystems that are individually tested before these sub-systems are themselves integrated to form the final system.
- System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

(b) Recovery Testing :

- Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic, re-initialisation, check pointing mechanisms, data recovery and restart are evaluated for correctness.
- If recovery requires human intervention, the Mean-Time-To-Repair (MTTR) is evaluated to determine whether it is within acceptable limits.
- Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed.
- Recovery testing should not be confused with reliability testing, which tries to discover the specific point at which failure occurs.

(c) Security Testing :

- Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.
- "The system's security must, of course, be tested for invulnerability from frontal attack but must also be tested for invulnerability from flank or rear attack."
- The security testing tasks include penetrating and destructive tests that are different from functional testing tasks.
- Systematic security testing approaches should be seamlessly incorporated into software engineering curricula and software development process.
- Security testing as a term has a number of different meanings and can be completed in a number of different ways :

(i) **Confidentiality :** A security measures which protects against the disclosure of information to parties other than the intended recipient that is by no means the only way of ensuring the security.

(ii) **Authentication :** This might involve confirming the identity of a person by tracing the origins of an artifact, ensuring that a product is what its packaging and labelling claims to be, or assuring that a computer program is a trusted one.

(iii) **Authorization :** The process of determining that a requester is allowed to receive a service or perform an operation. Access control is an example of authorization.

(iv) **Availability :** Assuring information and communications services will be ready for use when expected. Information must be kept available to authorized persons when they need it.

(d) Stress Testing :

- Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.

For example :

(i) Special tests may be designed that generate ten interrupts per second, when one or two is the average rate.

(ii) Input data rates may be increased by an order of magnitude to determine how input functions will respond.

(iii) Test cases that require maximum memory or other resources are executed.

(iv) Test cases that may cause thrashing in a virtual operating system are designed.

(v) Test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.

- Stress testing is the process of determining the ability of a computer, network, program or device to maintain a certain level of effectiveness under unfavorable conditions.
- The process can involve quantitative tests done in a lab, such as measuring the frequency of errors or system crashes. The term also refers to qualitative evaluation of factors such as availability or resistance to denial-of-service (DoS) attacks.
- The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources.
- Bugs and failure modes discovered under stress testing may or may not be repaired depending on the application, the failure mode, consequences, etc. The load in stress testing is often deliberately distorted so as to force the system into resource depletion.

(e) **Performance Testing :**

- Performance testing is designed to test the run-time performance of software within the context of an integrated system.
- Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as white-box tests are conducted.
- Once a system has been completely integrated, it is possible to test for emergent properties, such as performance and reliability. Performance tests have to be designed to ensure that the system can process its intended load.
- This usually involves running a series of tests where we increase the load until the system performance becomes unacceptable.
- Performance testing is defined as the technical investigation done to determine or validate the speed, scalability, and/or stability characteristics of the product under test.
- Performance-related activities, such as testing an tuning, are concerned with achieving response times, throughout, and resource-utilization levels that meet the performance objectives for the application under test.
- The following are the most common types of performance testing for web applications :

(i) **Performance test :** To determine or validate speed, scalability and stability.

(ii) **Load test :** To verify application behaviour under normal and peak load conditions.

(iii) **Stress test :** To determine or validate an application's behaviour when it is pushed beyond normal or peak load conditions.

(iv) **Capacity test :** To determine how many users and/or transactions a given system will support and still meet performance goals.

(f) **Functional Testing :**

- Functional testing is concerned with what the system does its features or functions. Non-functional testing is concerned with examining how well the system does. Non-functional testing like performance, usability, portability, maintainability, etc.
- Software requirement specification are appropriate at all levels of testing where a specification exists.
- For example, when performing system or acceptance testing, the requirements specification or functional specification may form the basis of the tests.

DEBUGGING

Q.28. Explain debugging software and its process. *CS : S-14(4M)*

Ans. Debugging :

- Debugging is the process of fixing errors and problems that have been discovered by testing. Using information from the program tests, debuggers use their knowledge of the programming language and the intended outcome of the test to locate and repair the program error.
- This process is often supported by interactive debugging tools that provide extra information about program execution.
- Debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code or the engineering of a hardware device.
- Debugging occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error. Although debugging can and should be an orderly process, it is still very much an art.
- A software engineer, evaluating the results of a test, is often confronted with a "symptomatic" indication of a software problem. That is, the external manifestation of the error and the internal cause of the error may have no obvious relationship to one another. The poorly understood mental process that connects a symptom to a cause is debugging.

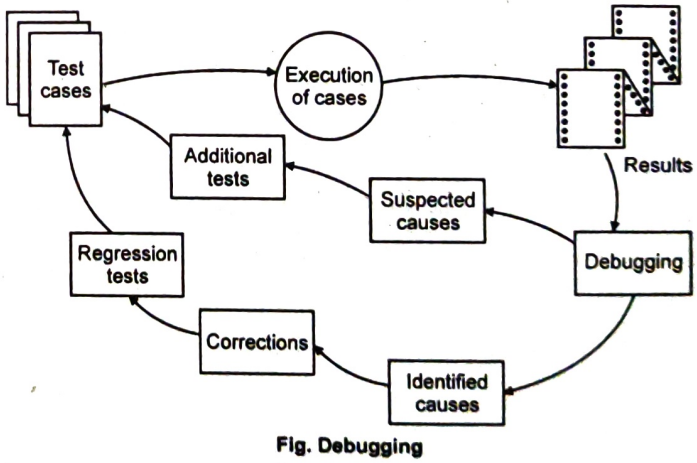


Fig. Debugging

Debugging process :

- 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 non corresponding data are a symptom of an underlying cause as yet hidden.

(5) The symptom may be due to causes that are distributed across a number of tasks running on different processors.

(6) The symptom may be a result of timing problems, rather than processing problems.

(7) It may be difficult to accurately reproduce input conditions.

(8) The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably.

POINTS TO REMEMBER :

(1) Testing is the process to detect the defects and minimize the risk associated with the residual defects.

(2) The activities that can be planned in advance and conducted systematically are known as testing strategy.

(3) Black-box testing is also known as functional testing or behavioural testing. It performs tests at software interface.

(4) White-box testing is also known as glass-box testing or structural testing which uses the control structure of the procedural design to derive test cases.

(5) Unit integration tests concentrate on functional verification of a component and incorporation of components into a program structure.

(6) Validation testing demonstrates traceability to software requirements.

(7) System testing validates software once it has been incorporated into a larger system.

(8) Regression testing is the activity that helps to ensure that changes, do not introduce unintended behaviour or additional errors.

(9) Smoke testing is an integration testing approach that is commonly used. When software products are being developed it is designed as a pacing mechanism for time-critical projects.

(10) Top-down integration testing is an incremental approach to construction of the software architecture.

(11) Bottom-up integration testing, as its name implies, begins construction and testing with atomic modules.

(12) Validation can be defined in many ways, but a simple definition is that validation succeeds when software function in a manner that can be reasonably expected by the customer.

(13) Alpha test is conducted at the developer's site by end-users.

(14) The beta test is conducted at end user sites.

(15) System testing is actually a series of different tests whose primary purpose is to fully exercise the computer - based system.

(16) Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.

(17) Security testing verifies that protection mechanism built into a system's security will, in fact, protect it from improper penetration.

(18) Stress testing executes a system in a manner that demands resources in abnormal quantity frequency or volume.

(19) Performance testing is often coupled with stress testing and usually requires both hardware and software instrumentation.

(20) Debugging is the process that results in the removal of the error.

(1) **Breakdown maintenance :**

It means that people waits until equipment fails and repair it. Such a thing could be used when the equipment failure does not significantly affect the operation or production or generate any significant loss other than repair cost.

(2) **Preventive maintenance :**

- It is a daily maintenance (cleaning, inspection, oiling and re-tightening), design to retain the healthy condition of equipment and prevent failure through the prevention of deterioration, periodic inspection or equipment condition diagnosis, to measure deterioration.
- It is further divided into periodic maintenance and predictive maintenance. Just like human life is extended by preventive medicine, the equipment service life can be prolonged by doing preventive maintenance.

(a) **Periodic maintenance (Time based maintenance - TBM) :**

- Time based maintenance consists of periodically inspecting, servicing and cleaning equipment and replacing parts to prevent sudden failure and process problems.

(b) **Predictive maintenance :**

- This is a method in which the service life of important part is predicted based on inspection or diagnosis, in order to use the parts to the limit of their service life.
- Compared to periodic maintenance, predictive maintenance is condition based maintenance. It manages trend values, by measuring and analyzing data about deterioration and employs a surveillance system, designed to monitor conditions through an on-line system.

(3) **Corrective maintenance :**

- It improves equipment and its components so that preventive maintenance can be carried out reliably.
- Equipment with design weakness must be redesigned to improve reliability or improving maintainability

(4) **Maintenance prevention :**

- It indicates the design of a new equipment.
- Weakness of current machines are sufficiently studied on site information leading to failure prevention, easier maintenance and prevention of defects, safety and ease of manufacturing and are incorporated before commissioning a new equipment.

PROJECT MANAGEMENT

Q.24. Define software process and project management. How the project plan help for the project management? CT: S-07(Old)

Ans. Software process :

- A software process provides the framework from which a comprehensive plan for software development can be established.
- A small number of framework activities are applicable to all software projects, regardless of their size or complexity.
- A number of different task sets tasks, milestones, work products and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
- Software process as a framework for the tasks that are required to build high-quality software.
- A software process defines the approach that is taken as software is engineered. But software engineering also encompasses technologies that populate the process technical methods and automated tools.

Project management :

- Project management involves the planning, monitoring and control of the people, process and events that occur as software evolves from a preliminary concept to an operational implementation.
- 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.
- The first activity in software project planning is the determination of software scope.
- Function and performance allocated to software during system engineering should be assessed to establish a project scope that is unambiguous and understandable at the management and technical levels.
- A statement of software scope must be bounded.
- Software scope describes the data and control to be processed, function, performance, constraints, interfaces and reliability.

Q.31. Write a note on W^5 HH principle. CS : W-11(2M)

Ans. **W^5 HH principle :**

- An approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches and required resource is called the WWWWWHH principle.
- Following are the series of questions that lead to a definition of the main characteristics of the project and the resulting project plan.

(1) **Why is the system being developed ?** : The answer to this question enables all parties to assess the validity of business reasons for the software work. Stated in another way, does the business purpose justify the expenditure of people, time and money.

(2) **What will be done, by when ?** : The answers to these questions help the team to establish a project schedule by identifying key project tasks and the milestones that are required by the customer.

(3) **Who is responsible for a function ?** : The role and responsibility of each member of the software team must be defined. The answer to this question helps to accomplish this.

(4) **Where are they organizationally located ?** : Not all roles and responsibilities reside within the software team itself. The customer, users and other stakeholders also have responsibilities.

(5) **How will the job be done technically and managerially ?** : Once product scope is established, a management and technical strategy for the project must be defined.

(6) **How much of each resource is needed ?** : The answer to this question is derived by developing estimates based on answers to earlier questions.

- Boehm's W^5 HH principle is applicable regardless of the size or complexity of a software project.

Q.32. Explain the steps in Software Project Planning.

Ans. In Software Project Planning following steps are performed :

(1) **Project Scope Definition and Scope Planning :** Every project needs a roadmap with clearly defined goals that should not change after the first phase of the project has been completed.

(2) **Develop a list of all deliverables :** Make sure all project team members are familiar with this list.

(3) **Time, Effort and Resource Estimation** : A document that clearly outlines all project milestones and activities required to complete the project should be created and maintained. Establish reasonable deadlines, taking into account project team members' productivity, availability and efficiency.

(4) **Create a budget for your project :** Ideally, project managers should be able to choose team members who work well together. Identify by name all individuals and/or organizations involved in the project. For each of them, roles and responsibilities on the project should be described in detail. Otherwise, miscommunication may occur leading to delays and situations where team members may have to redo their work.

(5) Identify the risks involved in the project and discuss alternatives if new requirements will be added to your project or members of your team will not meet the deadlines.

(6) Quality Planning.

(7) Risk Management Planning

(8) Set progress reporting guidelines - monthly, weekly or daily reports. Ideally, a collaborative workspace should be set up for the project online or offline where all parties can monitor the progress.

Q.33. Explain Software Scope.

OR What do you mean by 'statement of scope'? CT : W-08,11(8M)

OR Write a short note on software statement of scope.

CT : W-12(2M)

OR What is software scope? Justify with example, how functions, performance and constraints are evaluated together?

CT: S-13(7M)

Ans. **Software Scope :**

- Software scope describes the data and control to be processed, function, performance, constraints, interfaces and reliability.
- Functions described in the statement of scope are evaluated and in some cases refined to provide more detail prior to the beginning of estimation. Because both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful.
- Performance considerations encompass processing and response time requirements.
- Constraints identify limits placed on the software by external hardware, available memory or other existing systems.

RESOURCES

Q.35. **Write a note on resources.** **CT: W-12(2M)**

Q.36. **What are the different resources that can be used in developing any software?**

Ans. **Resources :**

- The development environment hardware and software tools sits at the foundation of the resources pyramid and provides the infrastructure to support the development effort.
- At a higher level, we encounter reusable software components software building blocks that can dramatically reduce development costs and accelerate delivery.

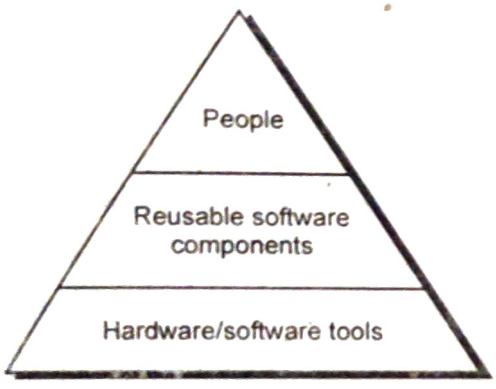


Fig. Project Resources

- At the top of the pyramid is the primary resource people. Each resource is specified with four characteristics :

(i) Description of the resource.

(ii) A statement of availability.

(iii) Time when the resource will be required.

(iv) Duration of time that resource will be applied.

- The last two characteristics can be viewed as a time window.
- Availability of the resource for a specified window must be established at the earliest practical time.

There are three types of resources :

(1) **Human resources :**

- The planner begins by evaluating scope and selecting the skills required to complete development.
- Both organizational position (e.g. manager, senior software engineer) and specialty (e.g. telecommunications, database, client/server) are specified.
- For relatively small projects (one person-year or less), a single individual may perform all software engineering tasks, consulting with specialists as required.
- The number of people required for a software project can be determined only after an estimate of development effort (e.g. person-months) is made.

(2) **Reusable software resources :**

- Component-based software engineering emphasizes reusability that is, the creation and reuse of software building blocks. Such building blocks, often called components, must be catalogued for easy reference, standardized for easy application and validated for easy integration.
- Bennatan suggests four software resources categories that should be considered as planning proceeds.

(i) **Off-the-shelf components :**

- Existing software that can be acquired from a third party or that has been developed internally for a past project.
- COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project and have been fully validated.

(ii) **Full-experience components :**

Existing specifications, designs, code or test data developed for past projects that are similar to the software to be built for the current project.

(iii) **Partial-experience components :**

- Existing specifications, designs, code or test data developed for past projects that are related to the software to be built for the current project but will require substantial modification.
- Members of the current software team have only limited experience in the application area represented by these components.

(iv) **Now components :**

Software components that must be built by the software team specifically for the needs of the current project.

(3) **Environmental Resources :**

- The environment that supports the software project, often called the software engineering environment (SEE), incorporates hardware and software.
- Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.

Example :

- The engineering and construction release activities are subdivided into the major software engineering tasks.
- Gross estimates of effort are provided for customer communication, planning, and risk analysis.
- These are noted in the total row at the bottom of the table. Horizontal and vertical totals provide an indication of estimated effort required for analysis, design, code and test.

DECOMPOSITION TECHNIQUES

Q.38. Explain Decomposition Technique.

OR Discuss various software decomposition techniques.

CT: S-14(6M)

OR What are the four approaches for sizing problem given by Putnam and Myres? *CS : S-10(4M), W-13(4M)*

Ans. Software decomposition techniques :

- Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e. developing a cost and effort estimate for a software project) is too complex to be considered in one piece.
- For this reason, we decompose the problem, re-characterizing it as a set of smaller problems.
- Estimation uses one or both forms of partitioning. But before an estimate can be made, the project planner must understand the scope of the software to be built and generate an estimate of its "size."

(1) Software Sizing :

- The accuracy of a software project estimate is predicated on a number of things as follows :

(i) The degree to which the planner has properly estimated the size of the product to be built.

(ii) The ability to translate the size estimate into human effort, calendar time and dollars (a function of the availability of reliable software metrics from past projects).

(iii) The degree to which the project plan reflects the abilities of the software team.

(iv) The stability of product requirements and the environment that supports the software engineering effort.

(2) "Fuzzy logic" sizing :

- This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic.
- To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale and then refine the magnitude within the original range.
- The planner should also have access to a historical database of projects, so that estimates can be compared to actual experience.

(3) Standard component sizing :

- Software is composed of a number of different "standard components" that are generic to a particular application area.
- For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC and object-level instructions.
- The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component

(4) Change sizing :

- This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project.
- The planner estimates the number and type (e.g. reuse, adding code, changing code, deleting code) of modifications that must be accomplished.
- Using an "effort ratio" for each type of change, the size of the change may be estimated.

Q.39. Explain LOC based estimation technique with example.

CT: W-06(4M), S-08(2M),CS : S-11(2M)

OR Explain LOC based estimation. Write the equations for different LOC-oriented estimation models. *CT: W-08(7M)*

Ans. LOC based estimation technique :

- A software package is to be developed for a computer-aided design application for mechanical components.
- A review of the system specification indicates that the software is to execute on an engineering workstation and must interface with

Q.55. How does decision tree help in making make-buy decision?

CT : S-06(7M), S-08(5M), W-08(3M), W-09,12(6M)

OR What is make-buy decision? Explain in brief with example.

CT : W-06,07, S-09(7M)

OR Explain the term "Make-Buy decision" with an example.

CT : S-07, W-11(6M)

OR Write a note on the make buy decision. **CT: W-10(5M)**

OR Write in brief about make buy decision tree. **CT : S-11(5M)**

OR Explain the term "Make-Buy Decision Tree" in context of in software development process. **CT: S-14(7M)**

Ans. Make-Buy Decision :

- In many software application areas, it is often more cost effective to acquire than develop computer software.
- Software engineering managers are faced with a make/buy decision that can be further complicated by a number of acquisition options :

(i) Software may be purchased (or licensed) off-the-shelf

(ii) "Fullexperience" or "partial-experience" software components may be acquired and then modified and integrated to meet specific needs.

(iii) Software may be custom built by an outside contractor to meet the purchaser's specifications.

- The steps involved in the acquisition of software are defined by the criticality of the software to be purchased and the end cost.
- For more expensive software products, the following guidelines can be applied :

(i) Develop specifications for function and performance of the desired software. Define measurable characteristics whenever possible.

(ii) Estimate the internal cost to develop and the delivery date.

(iii) Select three or four candidate applications that best meet our specifications.

(iv) Select reusable software components that will assist in constructing the required application.

(v) Develop a comparison matrix that presents a head-to-head comparison of key functions. Alternatively, conduct benchmark tests to compare candidate software.

(vi) Evaluate each software package or component based on past product quality, vendor support, product direction, reputation and the like.

(vii) Contact other users of the software and ask for opinions.

- The make/buy decision is made based on the following conditions :

(i) Will the delivery date of the software product be sooner than that for internally developed software?

(ii) Will the cost of acquisition plus the cost of customisation be less than the cost of developing the software internally?

(iii) Will the cost of outside support (e.g. a maintenance contract) be less than the cost of internal support?

- These conditions apply for each of the acquisition options

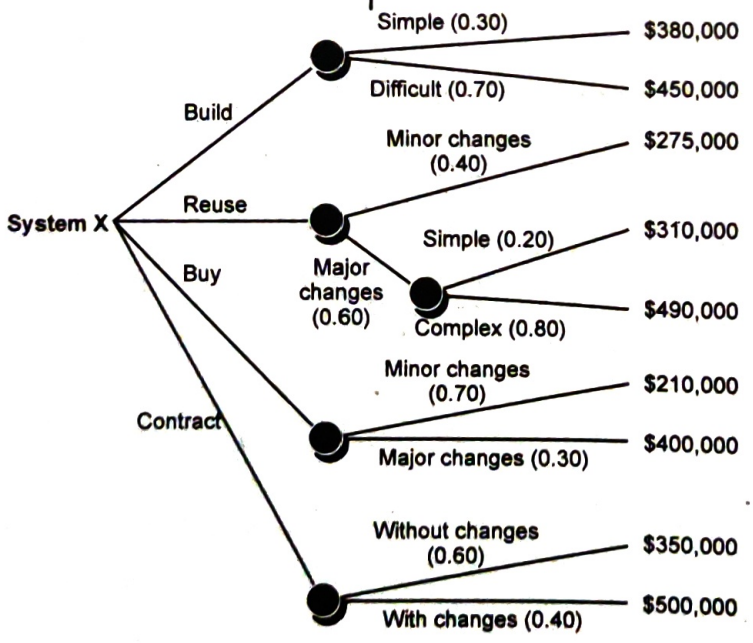


Fig. A decision tree to support the make/buy decision

- **For example :** Above figure depicts a decision tree for a software based system, X. In this case, the software engineering organization can

(1) Build system X from scratch.

(2) Reuse existing "partial-experience" components to construct the system.

(3) Buy an available software product and modify it to meet local needs.

(4) Contract the software development to an outside vendor.

- If the system is to be build from scratch, there is a 70 percent probability that the job will be difficult. Using the estimation techniques the project planner projects that a difficult development effort will cost $450000.
- A "simple" development effort is estimated to cost $380000. The expected value for cost, computed along any branch of the decision tree, is expected cost = (path probability) i × (estimated path cost) i where i is the decision tree path. For the build path, expected cost build = 0.30 ($380K) + 0.70 ($450K) = $429K.
- Following other paths of the decision tree, the projected costs for reuse, purchase and contract, under a variety of circumstances are also shown.
- The expected costs for these paths are expected.

Cost reuse = 0.40 ($275K) + 0.60 [0.20($310K) + 0.80($490K)]

= $382K

Expected cost buy = 0.70($210K) + 0.30($400K)] = $267K

Expected cost contract = 0.60($350K) + 0.40($500K)] = $410K

- Based on the probability and projected costs that have been noted in Fig the lowest expected cost is the "buy" option.
- It is important to note, however, that many criteria not just cost must be considered during the decision-making process.
- Availability experience of the developer / vendor / contractor, conformance to requirements, local "politics", and the likelihood of change are but a few of the criteria that may affect the ultimate decision to build, reuse, buy or contract.

METRICS FOR PROCESS AND PROJECT

Q.56. Define process metrics. *CT : S-10(2M)*

OR Explain process metrics and software process improvement. *CT : W-10(8M), W-13(6M)*

OR Explain process metrics. *CS: W-10(7M)*

Ans. Process metrics :

- The only rational way to improve any process is to measure specific attributes of the process, develop a set of meaningful metrics based on these attributes and then use the metrics to provide indicators that will lead to a strategy for improvement.
- Process sits at the center of a triangle connecting three factors that have a profound influence on software quality and organizational performance.

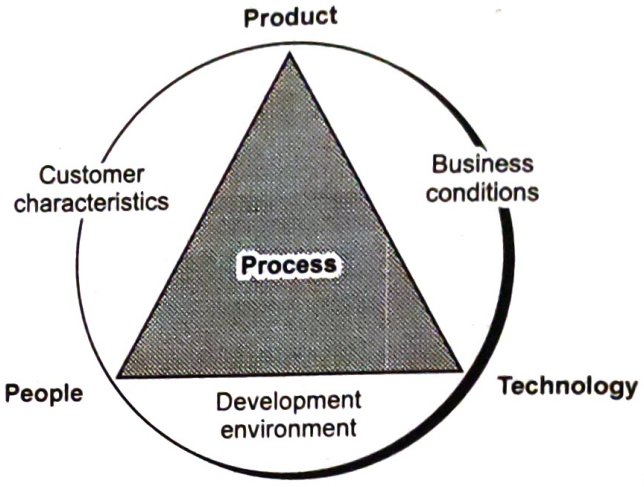


Fig : Determinants for software quality and organizational effectiveness

- The skill and motivation of people has been shown to be the single most influential factor in quality and performance.
- The complexity of the product can have a substantial impact on quality and team performance. The technology (i.e. the software engineering methods) that populate the process also has an impact.
- In addition, the process triangle exists within a circle of environmental conditions that include the development environment (e.g. CASE tools), business conditions (e.g. deadlines, business rules) and customer characteristics.
- Outcomes include measures of errors uncovered before release of the software, defects delivered to and reported by end-users, work products delivered (productivity), human effort expended, calendar time expended, schedule conformance and other measures.
- For example, we might measure the effort and time spent performing the umbrella activities and the generic software engineering activities.

Software process development :

- Software process improvement can and should begin at the individual level.
- Private process data can serve as an important driver as the individual software engineer works to improve.

SOFTWARE MEASUREMENT

Q.59. Write a detailed note on Software Measurement.

OR Explain the concept of software measurement. *CT : S-11(2M)*

OR What is software measurement? *CT : S-06, W-09(2M)*

Ans. Software measurement :

- Measurements in the physical world can be categorized in two ways : direct measures (e.g. the length of a bolt) and indirect measures (e.g. the "quality" of bolts produced, measured by counting rejects). Software metrics can be categorized similarly.
- Direct measures of the software engineering process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size and defects reported over some set period of time.
- Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability and many other "abilities".
- The cost and effort required building software, the number of lines of code produced, and other direct measures are relatively easy to collect, as long as specific conventions for measurement are established in advance. However, the quality and functionality of software or its efficiency or maintainability are more difficult to assess and can be measured only indirectly.
- We have already partitioned the software metrics domain into process, project and product metrics. We have also noted that product metrics that are private to an individual are often combined to develop project metrics that are public to a software team. Project metrics are then consolidated to create process metrics that are public to the software organization as a whole.
- To illustrate, we consider a simple example. Individuals on two different project teams record and categorize all errors that they find during the software process. Individual measures are then combined to develop team measures. Team A found 342 errors during the software process prior to release. Team B found 184 errors.
- Because we do not know the size or complexity of the projects, we cannot answer which team is more effective in uncovering errors. However, if the measures are normalized, it is possible to create software metrics that enable comparison in broader organizational averages.

Q.60. Explain size-oriented metric in details.

CT : S-06,11, W-06,07(3M), S-07, W-09(2M), S-09(6M), CS : W-08(5M), S-09(3M)

Ans. Size-oriented metrics :

- Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced.
- A table of size-oriented measures, such as the one shown in Figure :

Project	LOC	Effort	$(000)	Pp.doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
.	.	.	.	.	.		
.	.	.	.	.	.		
.	.	.	.	.	.		

Fig : Size-oriented metrics

- The table entry for project alpha : 12,100 lines of code were developed with 24 person-months of effort at a cost of $168,000.
- It shows that the effort and cost recorded in the table represent all software engineering activities (analysis, design, code and test), not just coding.
- Information for project alpha indicates that 365 pages of documentation were developed, 134 errors were recorded before the software was released and 29 defects were encountered after release to the customer within the first year of operation.
- In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value.
- The above data contained in the table, a set of simple size-oriented metrics can be developed for each project :

  (i) Errors per KLOC (thousand lines of code).

  (ii) Defects per KLOC.

  (iii) $ per LOC.

  (iv) Page of documentation per KLOC.
- In addition, other interesting metrics can be computed :

  (i) Errors per person-month.

  (ii) LOC per person-month.

  (iii) $ per page of documentation.

METRICS FOR SOFTWARE QUALITY

Q.69. Explain the Metrics for Software Quality.

Ans. **The Metrics for Software Quality :**

- The goal of software engineering is to produce a high-quality system, application or product. To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process.
- The quality of a system, application or product is only as good as the requirements that describe the problem, the design that models the solution.
- The code that leads to an executable program and the tests that exercise the software to uncover errors.
- Private metrics collected by individual software engineers are assimilated to provide project level results.
- Many quality measures can be collected, the primary thrust at the project level is to measure errors and defects.
- Metrics such as work product (e.g. requirements or design) errors per function point, errors uncovered per review hour and errors uncovered per testing hour provide insight into the efficacy of each of the activities implied by the metric.
- Error data can also be used to compute the defect removal efficiency (DRE) for each process framework activity.

Q.70. What are the different measures for measuring software quality? Discuss them in detail. *CT : W-08(6M), CS : W-08(2M)*

OR **State and explain measures of software quality.** *CS : S-10(4M)*

CT : S-06,09, W-07,12(7M), W-10(6M), W-11(4M)

OR **Explain the factors by which quality is measured in software development.** *CT : W-09(6M)*

Ans. **The different measures for measuring software quality are as follows :**

(1) Correctness :

- A program must operate correctly or it provides little value to its users.
- Correctness is the degree to which the software performs its required function.
- The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements.
- When considering the overall quality of a software product, defects are those problems reported by a user of the program after the program has been released for general use.
- For quality assessment purposes, defects are counted over a standard period of time, typically one year.

(2) Maintainability :

- Software maintenance accounts for more effort than any other software engineering activity.
- Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes or enhanced if the customer desires a change in requirements.
- There is no way to measure maintainability directly, therefore, we must use indirect measures.
- A simple time-oriented metric is mean-time-to-change (MTTC), the time it takes to analyze the change request, design an appropriate modification, implement the change, test it and distribute the change to all users.

(3) Integrity :

- Software integrity has become increasingly important in the age of hackers and firewalls.
- This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security.
- Attacks can be made on all three components of software : programs, data and documents.
- To measure integrity, two additional attributes must be defined that is threat and security.
- Threat is the probability (which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time.
- Security is the probability (which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled.
- The integrity of a system can then be defined as :

  Integrity = summation [(1 – threat) * (1 – security)]

  where threat and security are summed over each type of attack.

(4) Usability :

- Usability is an attempt to quantify user-friendliness. It can be measured in terms of following characteristics :

(ii) The physical and /or intellectual skill required to learn the system.

(iii) The time required to become moderately efficient in the use of the system.

(iv) The net increase in productivity (over the approach that the system replaces) measured when the system is used by someone who is moderately efficient.

(v) A subjective assessment (sometimes obtained through a questionnaire) of users attitudes toward the system.

Q.71. Write a note on Defect removal efficiency.

CT-W-10(4M), W-12(2M)

OR What is DRE? Explain with respect to project as a whole and DRE within the project. *CT : S-12(6M)*

Ans. Defect Removal Efficiency :

- A quality metric that provides benefit at both the project and process level is defect removal efficiency (DRE).
- In essence, DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.
- When considered for a project as a whole, DRE is defined in the following manner :

  DRE ⇒E/(E + D)

  where E is the number of errors found before delivery of the software to the end-user and D is the number of defects found after delivery.
- The ideal value for DRE is 1. That is, no defects are found in the software.
- D will be greater than 0, but the value of DRE can still approach 1.
- As E increases (for a given value of D), the overall value of DRE begins to approach 1. In fact, as E increases, it is likely that the final value of D will decrease.
- DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task.
- When used in this context, we redefine DRE as

  $DRE_i = E_i /(E_i + E_{i+1})$

  where E_i is the number of errors found during software engineering activity i and E_{i+1} is the number of errors found during software engineering activity i+1 that are traceable to errors that were not discovered in software engineering activity i.

PROJECT SCHEDULING

Q.72. Explain the concept of scheduling. *CS : W-11(4M)*

Ans. Scheduling :

- Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort.
- Therefore, generalized project scheduling tools and techniques can be applied with little modification to software projects.
- Program evaluation and review technique (PERT) and critical path method (CPM) are two project scheduling methods that can be applied to software development.
- Both techniques are driven by information already developed in earlier project planning activities in following areas :

  (i) Estimates of effort.

  (ii) A decomposition of the product function.

  (iii) The selection of the appropriate process model and task set.

  (iv) Decomposition of tasks.
- Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project work breakdown structure (WBS), are defined for the product as a whole or for individual functions.
- Both PERT and CPM provide quantitative tools that allow the software planner as follows :

(i) Determine the critical path-the chain of tasks that determines the duration of the project.

(ii) Establish "most likely" time estimates for individual tasks by applying statistical models.

(iii) Calculate "boundary times" that define a time "window" for a particular task.

- Important boundary times that may be discerned from a PERT or CPM network are :

(i) The earliest time that a task can begin when all preceding tasks are completed in the shortest possible time.

(ii) The latest time for task initiation before the minimum project completion time is delayed.

(iii) The earliest finish the sum of the earliest start and the task duration.

(5) **Control :**

- Risk control correct deviations from planned risk actions.
- Risk control is a part of project management and relies on project management processes to control risk action plans, correct for variations from plans, respond to triggering events and improve risk management processes.

(6) **Communicate :**

- Communication lies at the center of the paradigm in order to emphasize its pervasiveness and its criticality.
- Without effective communication, no risk management approach can be viable. It is an integral part of all the other risk management activities.
- Clearly, professionals associated with a project are the most qualified to identify risk in their work on a daily basis.
- One should ask, "Does project management provide a conducive environment for people to share their concerns regarding potential risks?

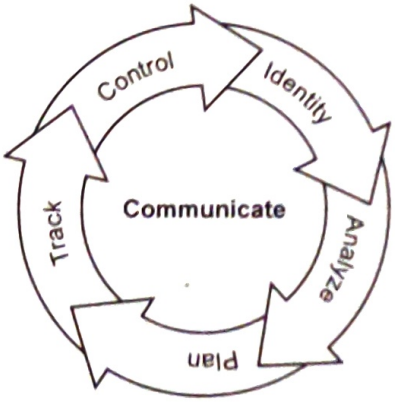


Fig. Risk management model

RISK STRATEGIES

Q.4. Explain reactive and proactive risk strategies.

Ans.

(1) Reactive risk strategies :

- In case of reactive risk strategies the reactive management refers to a situation in which we can't or don't plan ahead for problems or opportunities and we react to them as they happen.
- The majority of software teams rely solely on reactive risk strategies.
- The reactive strategy monitors project for likely risks.
- Resources are set aside to deal with them, should they become actual problems.
- The software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a fire fighting mode.
- When this fails "crisis management" takes over and the project is in real jeopardy.

(2) Proactive risk strategies :

- In case of proactive risk strategies the proactive management happens when we plan ahead to avoid or manage problems.
- The most intelligent strategy for risk management is to be proactive. A proactive strategy begins before the technical work is initiated.
- Potential risks are identified, their probability and impact are assessed and they are ranked by importance. Then, the software team establishes a plan for managing risk.
- The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner.

SOFTWARE RISKS

Q.5. What is risk? ***CT : W-08(1M), S-10(2M), CS : W-08(2M)***

OR What is risk? What are different types of risks? ***CT : W-10(3M)***

OR What is risk? How to identify the risk? ***CS : W-10(4M)***

OR Write a short note on software risks. ***CT : W-11(2M)***

OR List various types of risks in software computing.

CS : S-10(3M), W-12(4M)

OR Define a risk. ***CT : W-13(1M)***

Ans. Software risk:

- Software risk is defined as a general agreement in which risk always involves two characteristics.

(i) **Uncertainty :** In uncertainty, the risk may or may not happen. There are no 100% probable risks.

(ii) **Loss :** If the risk becomes a reality, unwanted consequences or losses will occur.

- It is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered as follows :

(1) Project risks :

- Project risk threaten the projects plans.

- If project risks become real, it is likely that project schedule will slip and that costs will increase.
- Project risks identify potential budgetary, schedule, personnel, resource, customer and requirements problems and their impact on a software project.

(2) **Technical risks :**

- Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible.
- Technical risks identify potential design, implementation, interface, verification and maintenance problems.

(3) **Business risks :**

- Business risks threaten the viability of the software to be built. The top five business risks are :

(a) Building an excellent product or system that no one really wants.

(b) Building a product that no longer fits into the overall business strategy for the company.

(c) Building a product that the sales force doesn't understand how to sell.

(d) Losing the support of senior management due to a change in focus or a change in people.

(e) Losing budgetary or personnel commitment (budget risks). Some risks are simply unpredictable in advance.

(4) **Product risks :**

- Risks that affect the quality or performance of the software being developed are known as the product risks.
- An example of a product risk is the failure of a purchased component to perform as expected. This may affect the overall performance of the system so that it is slower than expected.

(5) **Known risks :**

- Known risks are those that can be uncovered after careful evaluation of project plan, the business and technical environment in which the project is being developed and other reliable information sources.
- For example, unrealistic delivery date, lack of documented requirements or software scope, poor development environment, etc.

(6) **Predictable risks :**

- These are extrapolated from past project experience.
- For example, staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance are serviced.

(7) **Unpredictable risks :**

- These are the joker in the deck.
- They can and do occur, but they are extremely difficult to identify in advance.

Q.6. Discuss the various activities involved in risk analysis.

OR What is risk analysis?

Ans. Risk analysis :

- Analysis is the conversion of risk data into risk decision-making information. Analysis is reviewing, prioritizing and selecting most critical risks to work on.
- Three main steps can be used to analyze software risks. They are :

(1) **Probability :** First, a probability of the occurrence of a particular risk is estimated.

(2) **Impact :** The impact to the project of the particular risk should be estimated.

(3) **Overall risk :**

- The last step in analysis is to determine the overall risk to the project.
- Using estimates on risk probabilities and impacts, the overall risk to the project should be gauged.
- A matrix should used to be determine overall risk for each of effort, performance and schedule. One such matrix, is given in the table below :

Impact/ Probability	Very High	High	Medium	Low	Very Low
Catastrophic	High	High	Moderate	Moderate	Low
Critical	High	High	Moderate	Low	None
Marginal	Moderate	Moderate	Low	None	None
Negligible	Moderate	Low	Low	None	None

Fig. Impact/Probability Matrix

- For example, If a risk has a high probability and a Marginal impact to performance, the overall risk to performance would be Moderate. If the overall risk for both effort and schedule are also Moderate, the overall risk to the project would be classified as Moderate.

- Once the analysis is complete, a certain number of the highest overall impact risks should be selected for planning. Whether this number is two, five, ten or twenty depends on the overall risk and the wishes of the instructor.
- For those risks whose overall impact is Low or none, it makes little sense to squander resources in the planning stage.
- Various risk analysis methods/techniques are as follows :

(a) Upside risk :

(i) Market survey.

(ii) Prospecting.

(iii) Test marketing.

(iv) Research and development.

(v) Business impact analysis.

(b) Both :

(i) Dependency modeling.

(ii) SWOT analysis (Strengths, Weaknesses, Opportunities, Threats).

(iii) Business continuity planning.

(iv) Event tree analysis.

(v) BPEST (Business, Political, Economic, Social, Technological) analysis.

(vi) Real option modeling.

(vii) Decision taking under conditions of risk and uncertainty.

(viii) Statistical inference.

(ix) Measures of central tendency and dispersion.

(x) PESTLE (Political Economic Social Technical Legal Environmental).

(c) Downside risk :

(i) Threat analysis.

(ii) Fault tree analysis.

(iii) FMEA (Failure Mode and Effect Analysis).

Q.7. Write note on risk analysis and management. *CT : S-07(3M)*

Ans. Risk analysis and management :

- Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty.
- Many problems can plague a software project. A risk is a potential problem it might happen, it might not.
- But, regardless of the outcome, it's a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan should the problem actually occur.
- When risk is considered in the context of software engineering, The three conceptual underpinnings are always in evidence.
- The future is our concern what risks might cause the software project to go awry.
- Change is our concern how will changes in customer requirements, development technologies, target computers, and all other entities connected to the project affect timeliness and overall success.

RISK IDENTIFICATION

Q.8. Explain Risk Identification.

CT : S-06,08(3M), W-10(10M), W-13(5M)

OR Write short note on risk identification.

CT : S-07(3M), W-11,12(3M)

Ans. Risk Identification :

- Risk identification is the first stage of the risk management process. It is concerned with identifying the risks that could pose a major threat to the software engineering process, the software being developed or the development organization.
- During this first step in the software risk management process, risks are identified and added to the list of known risks. The output of this step is a list of project specific risks that have the potential of compromising the project's success.
- By identifying predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.
- In risk identification there are two distinct types of risks that are generic risks and product specific risks.
- Generic risks are a potential threat to every software project. Product-specific risks can be identified only by those with a clear understanding of the technology and the environment that is specific to the project in hand.
- There are many techniques for identifying risks, including checklists, interviewing, reporting, decomposition, assumption analysis, critical path analysis and utilization of risk taxonomies. These are explained as follows :

(1) **Checklists :**

- One of the most popular method for identifying risk is to create a risk item checklist. Checklist can be used for risk identification and focuses on some subset of known and predictable risk.
- Risk item checklist can be organized in different and various ways. Questions relevant to each of the topics can be answered for each software project. Answers to these questions allows the planner to estimate the impact of risk.
- A different and various risk item checklist format that are relevant to each of the generic subcategories are as follows :

(i) **Process definition:** Risks associated with the degree to which the software process has been defined and is followed by the development organization.

(ii) **Development environment :** Risks associated with the availability and quality of the tools to be used to build the product.

(iii) **Technology to be built :** Risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.

(iv) **Staff size and experience:** Risks associated with the overall technical and project experience of the software engineers who will do the work.

(v) **Product size :** Risks associated with the overall size of the software to be built.

(2) **Interviewing/Brainstorming :**

- One technique for identifying risks is interviewing or brainstorming with project of personnel, customers and vendors.
- It can be a team process where a team get together to brainstorm possible risks.
- The project manager may simply use his or her experience to identify the most probable or critical risks.
- It is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.).
- Open ended questions such as the following can help to identify potential areas of risk :

(i) What new or improved technologies does this project implement?

(ii) What interfaces issues still need to be defined?

(iii) What requirements exist that we aren't sure how to implement?

(iv) What concerns do we have about our ability to meet the required quality and performance levels?

(3) **Voluntary reporting :**

- Another risk identification technique is voluntary reporting, where any individual who identifies a risk is encouraged and rewarded for bringing that risk to management's attention. This requires the complete elimination of the "shoot the messenger" syndrome.
- It avoids the temptation to assign risk reduction actions to the person who identified the risk. Risks can also be identified through required reporting mechanisms such as status reports or project reviews.

(4) **Decomposition :**

- As the product is being decomposed during the requirements and design phases, another opportunity exists for risk identifications. Every TBD (To Be One/Determined) is a potential risk.
- As Ould states, "The most important thing about planning is writing down what you don't know, because what you don't know is what you must find out."
- Decomposition in the form of work breakdown structures during project planning can also help to identify areas of uncertainty that may need to be recorded as risks.

(5) **Assumption analysis :**

- Process and product assumptions must be analyzed.
- For example, We might assume the hardware would be available by the system test date or three additional experienced C++ programmers will be hired by the time coding starts. If these assumptions prove to be false, we could have major problems.

(6) **Critical path analysis :**

- As we perform critical path analysis for our project plan, we must remain on the alert to identify risks.
- Any possibility of schedule slippage on the critical path must be considered a risk because it directly impacts our ability to meet schedule.

(7) **Risk taxonomies ;**

- Risk taxonomies are lists of problems that have occurred on other projects and can be used as checklists to help ensure all potential risks have been considered.
- An example of a risk taxonomy can be found in the Software Engineering Institute's Taxonomy Based Risk Identification report that covers thirteen major risk areas.

Q.9. Explain risk projection. CT : S-06,08(3M), CS : W-10(3M)

OR Write short note on risk projection.

CT : W-08,11(3M), S-10(5M), CS : W-08(6M)

OR Explain projection of risk in the view of risk management.

CT : S-12(7M)

Ans. Risk projection :

- It is also called risk estimation. The attempts to rate each risk should occur in two ways : the likelihood or probability that the risk is real and the consequences of the problems associated with the risk.
- The project planner, along with other managers and technical staff, performs four risk projection activities.

(i) Establish a scale that reflects the perceived likelihood of a risk.

(ii) Delineate the consequences of the risk.

(iii) Estimate the impact of the risk on the project and the product.

(iv) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing a Risk Table :

- It provides a project manager with a simple technique for risk projection. A sample risk table is shown as follows : A project team begins by listing all risks in the first column of the table.
- This can be accomplished with the help of the risk item checklists. Each risk is categorized in the second column.
- The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually.

Table : Sample Risk process sorting.

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low.	PS	60%	2	
Larger number of users than planned.	PS	30%	3	
Less reuse than planned.	PS	70%	2	
End-users resist system.	BU	40%	3	
Delivery deadline will be tightened.	BU	50%	2	
Funding will be lost.	CU	40%	1	
Customer will change requirements.	PS	80%	2	
Technology will not meet expectations.	TE	30%	1	
Lack of training on tools.	DE	80%	3	
Staff experienced.	ST	30%	2	
Staff turnover will be high.	ST	60%	2	
⋮	⋮	⋮	⋮	

Impact values :

1 : Catastrophic.

2 : Critical.

3 : Marginal.

4 : Negligible.

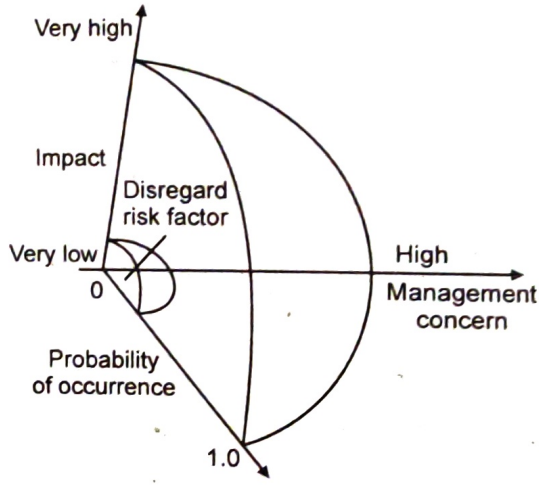


Fig. Risk and management concern

- The project manager studies the resultant sorted table and defines a cutoff line. The cut-off line implies that only risks that lie above the line will be given further attention. Risks that fall below the line are re-evaluated to accomplish second order prioritization.
- Referring to Fig. risk impact and probability have a distinct influence on management concern. A risk factor that has a high impact but of a very low probability of occurrence should not absorb a significant amount of management time. However, high-impact risks with moderate to high probability and low-impact risk with high probability should be carried forward into the risk analysis steps that follow.

- The column named RMMM contains a pointer into a Risk Mitigation, Monitoring and Management Plan or alternatively, a collection of risk information sheets developed for all risks that lie above the cut-off.

Q.10. Explain the term assessing risk impact.

Ans. Assessing risk impact :

- The parameter which affect the consequences that are likely if a risk occurs are its nature, its scope, and its timing.
- The nature of the risk indicates the problems that are likely if it occurs.
- For example, a poorly defined external interface to customer hardware will preclude early design and testing and will likely lead to system integration problems late in a project.
- The scope of a risk combines the severity with its overall distribution.
- The timing of a risk considers when and for how long the impact will be felt.
- In the risk analysis approach proposed by the U.S. Air Force. The following steps are recommended to determine the overall consequences of a risk :

(i) Determine the average probability of occurrence value for each risk component.

(ii) Determine the impact for each component based on the criteria .

(iii) Complete the risk table and analyze the results.

- The overall risk exposure, RE is determined using the following relationship :

$RE = P \times C$

where, P is the probability of occurrence for a risk and C is the cost to the project the risk occur.

For example, assume that the software term defines a project risk in the following manner:

Risk Identification : Only 70 percent of the software components scheduled for reuse. The remaining functionality will have to be custom developed.

Risk probability : 80% (likely).

Risk impact : 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch. Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $ 14,00, the overall cost (impact) to develop the components would be

$18 \times 100 \times 14 = \$ 25,200.$

Risk exposure : Risk exposure can be computed for each risk in the risk table, once an estimate of the cost of the risk is made. The total risk exposure for all risks can provide a means for adjusting the final cost estimate for a project.

$\therefore$ R.E. $= 0.80 \times 25,200 \sim \$ 20,200$

Q.11. Explain 'risk assessment'. *CT : S-06,08(3M), W-11(3M)*

Ans. Risk assessment :

- In the risk management process, a set of triplets of the form $[r_i, l_i, x_i]$ is established.

  Where, r_i is risk, l_i is the likelihood (probability) of the risk and x_i is the impact of the risk.
- A risk referent level must be defined. For most software projects, performance, cost, support and schedule also represent risk referent levels. That is, there is a level for performance degradation, cost overrun, support difficulty or schedule slippage that will cause the project to be terminated.
- In the context of software risk analysis, a risk referent level has a single point, called the referent point or break point, at which the decision to proceed with the project or terminate it are equally weighted. If a combination of risks create problem that cause one or more of these referent levels to be exceeded, work will stop.

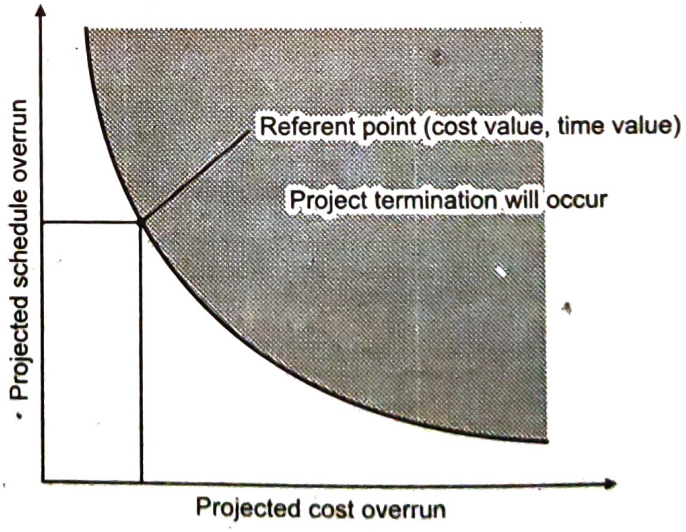


Fig. Risk refine level

OR What is RMMM? CS : S-10(2M)

OR Explain the concept of RMMM. CS : W-12(4M)

OR Write short note on RMMM. CT : S-09(6M)

Ans.

(1) Risk Mitigation :

- When the software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation.
- For example, assume that high staff turnover is noted as a project risk, r1. Based on past history and management intuition, the likelihood of high turnover is estimated to be 0.70 (70 percent, rather high) and the impact, x1 is projected at level 2. That is high turnover will have a critical impact on project cost and schedule.
- To mitigate this risk, project management must develop a strategy for reducing turnover. Among the possible steps to be taken are meet with current staff to determine causes for turnover.
- Mitigate those causes that are under our control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.
- Assign a backup staff member for every critical technologist.

(2) Risk Monitoring :

- When the project proceeds, risk monitoring activities commence.
- The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely.
- In the case of high staff turnover, the following factors can be monitored:

(i) General attitude of team members based on project pressures.

(ii) The degree to which the team has jelled.

(iii) Interpersonal relationships among team members.

(iv) Potential problems with compensation and benefits.

(v) The availability of jobs within the company and outside it.

(vi) In addition to monitoring these factors, the project manager should monitor the effectiveness of risk mitigation steps.

(3) Risk management :

- Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.
- If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team.
- In addition, the project manager may temporarily refocus resources to those functions that are fully staffed, enabling newcomers who must be added to the team to "get up to speed."
- It is important to note that RMMM steps incur additional project cost. For example, spending the time to "backup" every critical technologist costs money.
- Part of risk management is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them.
- In essence, the project planner performs a classic cost/benefit analysis.
- If risk aversion steps for high turnover will increase both project cost and duration by an estimated 15 percent, but the predominant cost factor is "backup" management may decide not to implement this step.
- For a large project, 30 or 40 risks may identified. If between three and seven risk management steps are identified for each, risk management may become a project in itself. For this reason, we adapt the Pareto 80–20 rule to software risk.

Q.15. What are the safety risks and hazards associated with software project?

Ans. Safety risks and hazards :

- Risks can be occur after the software has been successfully developed and delivered to the customer.
- It is associated with the consequences of software failure in the field.
- In early days of computing, although the probability of failure of a well-engineered system was small, an undetected fault in a computer based control or monitoring system could result in enormous economic damage or worse, significant human injury. Today,

computer hardware and software are uses regularly to control safety critical systems.

- When software is used as part of a control system, complexity can increase by an order of magnitude or more. Subtle design faults induced by human error something that can be uncovered and eliminated in hardware-based conventional control become much more difficult to uncover when software is used.
- Software safety and hazard analysis are software quality assurance activities that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
- If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.

Q.16. For a software company high staff turn over is noted as project risk. Explain RMMM plan to handle this uncertainty.

CT : S-13(7M)

OR Write note on RMMM plan. *CT : S-14(4M), CS : S-13(6M)*

Ans. RMMM Plan :

- The risk management plan can be included in the software project plan or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan.
- In the RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
- Some software teams do not develop a formal RMMM document.
- Each risk is documented individually using a Risk Information Sheet (RIS).
- When RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence.
- Risk mitigation is a problem avoidance activity.
- Risk monitoring is a project tracking activity following three primary objectives:

(i) To assess whether predicted risks do, in fact, occur.

(ii) To ensure that risk aversion steps defined for the risk are being properly applied.

(iii) To collect information that can be used for future risk analysis. In many cases, the problems that occur during a project can be traced to more than one risk.

For example, a risk information sheet is shown below :

Risk information sheet			
Risk ID : PO2-432	Date : 5/9/02	Prob. : 80%	Impact high
Description : Only 70 percent of the software component scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context : Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2 : The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring : (1) Contact third party to determine conformance with design standards. (2) Press for interface standards completion; consider component structure when deciding on interface protocol. (3) Check to determine number of components in sub condition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger : RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom build; allocate staff accordingly. Trigger : Mitigation step unproductive as on 7/1/02.			
Current status : 5/12/02 : Mitigation steps initiated.			
Originator : D. Gagne		Assigned : B. Laster	

Fig. A risk information sheet

Q.17. What are different issues in risk management?

CT : W-06,07(6M), W-12(7M)

Ans. Issues in risk management :

- Risk management in projects involves identifying, quantifying, and managing risks. All projects have some measure of risk. Projects using new technology face the prospect of that technology failing to deliver on expectations highly complex projects deal with the problem of being able to accurately estimate time and costs and even the smallest and simplest projects have some element of risk.
- It is impossible to remove all risks, so we try to identify and manage them to prevent project failure. A risk plan is the only way to obtain project approval, as it presents the risks as well-defined and, therefore, controllable.
- Risk management is considered a major part of the project management process, but can it help with such events? And if it can't, why do we expend time and energy trying to predict and control the unpredictable?
- Many risk management plans are little more than a standard template that lists the same risk factors for every project un-documented assumptions, failure to estimate tasks accurately, key team members re-assigned, etc. Surely by now we all know that these uncertainties exist in every project.
- If we know about potential risks, why are we even calling them risks aren't they simply the inherent uncertainties present in doing anything new?
- Another problem with risk management is that many potential risks are predicted by past events, yet any stockbroker will tell you that you cannot predict future financial markets by looking at historic trends. The best that we are doing is guessing until a risk event actually occurs, we cannot say with any certainty that it will happen.
- Risk management may seem like a sensible process, and maybe it is useful for a novice project manager, but it fails to effectively address the real project risks those factors that cannot be anticipated.

QUALITY MANAGEMENT

Q.18. Explain the term quality management.

OR What is quality management?

OR Explain software quality Management. CS : W-08(3M)

Ans. Quality management :

- Quality management encompasses the following:

(1) A software quality assurance (SQA) process.

(2) Specific quality assurance and quality control tasks (including formal technical reviews and a multi tiered testing strategy).

(3) Effective software engineering practice (methods and tools).

(4) Control of all software work products and the changes made to them.

(5) A procedure to ensure compliance with software development standards (when applicable).

(6) Measurement and reporting mechanisms.

- We focus on the management issues and the process-specific activities that enable a software organization to ensure that it does the right things at right time in the right way.
- Variation control is the heart of quality control.
- From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment and calendar time.
- In general, we would like to make sure that our testing program covers a known percentage of the software, from one release to another. Not only we want to minimize the number of defects that are released to the field but also, we'd like to ensure that the variance in the number of bugs is minimized from one release to another.

QUALITY CONCEPTS

Q.19. Define quality.What are its types?

Ans. Quality :

- The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations is known as quality.
- Quality can also be defined as a characteristic or attribute of something. As an attribute of an item, quality refers to measurable characteristics things we are able to compare to known standards such as length, color, electrical properties and malleability.

There are two types of quality :

Table : Data collection for statistical SQA

	Total		Serious		Moderate		Minor	
Error	No.	%	No.	%	No.	%	No.	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Total	**942**	**100%**	**128**	**100**	**379**	**100%**	**435**	**100%**

S_i = The number of serious errors.

M_i = The number of moderate errors.

T_i = The number of minor errors.

PS = Size of the product at the i^{th} step.

w_s, w_m, w_t = Weighting factors for serious, moderate and trivial errors, where recommended values are $w_s = 10$, $w_m = 3$, $w_t = 1$.

- The weighting factors for each phase should become larger as development progresses. This rewards an organization that finds errors early.
- At each step in the software process, a phase index, PI_i is computed.

$$PI_i = w_s\left(\frac{S_i}{E_i}\right) + w_m\left(\frac{M_i}{E_i}\right) + w_t\left(\frac{T_i}{E_i}\right)$$

- The error index is computed by calculating the cumulative effect on each PI_i weighting encountered later in the software engineering process more heavily than those encountered earlier.

$$EI = \Sigma(i \times PI_i)/PS$$

$$= (PI_1 + 2PI_2 + 3PI_3 + ... + iPI_i)/PS$$

SOFTWARE RELIABILITY

Q.34. Write short note on software reliability.

CT : S-07(3M), W-09(4M), W-10(5M)

CS : W-08(5M), W-09(7M), W-10(3M)

OR What are different factors affecting reliability?

CT : W-06(5M), S-08(6M)

OR Define software reliability and availability. Give measures of it.

CS : S-09(5M), CT : W-08(7M)

OR How can we measure software reliability and availability?

CS : S-12(4M)

OR Comment on "software reliability and its measures".

CT : S-13(6M)

OR What is reliability and availability? What are the measures of reliability and availability? *CS : W-13(5M)*

Ans. Software reliability and availability:

- The probability of failure-free software operation of a computer for a specified period of time in a specified environment is known as software reliability.
- Software reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection.
- The high complexity of software is the major contributing factor of software reliability problems.
- Software reliability is not a function of time although researchers have come up with models relating the two.
- The modeling technique for software reliability is reaching its prosperity, but before using the technique, we must carefully select the appropriate model that can best suited to our case. Measurement in software is still in its infancy.
- Software are availability means a probability that a program is operating according to the requirements at a given point of time.
- Various approaches can be used to improve the reliability of software, it is hard to balance development time and budget with software reliability.

Measures of reliability and availability :

- Reliability is a measure of the probability that an item will perform its intended function for a specified interval under stated conditions.
- Reliability measures are generally used to calculate the probability of failure for a single solution component. One measure used to define a component or system's reliability is mean time between failures (MTBF).
- Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects. In hardware, failures due to physical wear (e.g. the effects of temperature, corrosion, shock) are more likely than a design-related failure. Concepts apply to both system elements.
- If we consider a computer-based system, a simple measure of reliability is meantime between-failure (MTBF).

  Where, MTBF = MTTF + MTTR

  MTTF and MTTR are mean time to failure and mean time to repair, respectively.
- MTBF is a far more useful measure than defects/KLOC or defects/FP. Stated simply, an end-user is concerned with failures, not with the total error count.
- For example, consider a program that has been in operation for 14 months. Many errors in this program may remain undetected for decades before they are discovered.
- In addition to a reliability measure, we must develop a measure of availability. Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as :

  Availability = [MTTF/(MTTF + MTTR)] x 100%
- The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR and is an indirect measure of the maintainability of software.

Q.35. Write a note on software safety.

Ans. Software safety :

- Software safety is directly related to the more critical design aspects and safety attributes in software and system functionality, whereas software quality attributes are inherently different and require standard scrutiny.
- Software safety hazard analysis required for more complex systems where software is controlling critical functions generally in the following sequential categories and are conducted in phases as part of the system safety or safety engineering to process.
- Software safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
- If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.
- Software reliability and software safety are closely related to one another, it is important to understand the subtle difference between them.
- Software reliability uses statistical analysis to determine the likelihood that a software failure will occur.
- Software safety examines the ways in which failures result in conditions that can lead to a mishap. That is, failures are not considered in a vacuum, but are evaluated in the context of an entire computer-based system.
- A modeling and analysis process is conducted as a part of software safety hazards are identified and categorized by criticality and risk.
- Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence.
- Once hazards are analyzed, safety-related requirements are specified for the software. That is, the specification can contain a list of events and the desired system responses to these events. The role of managing undesirable events is then indicated.

CHANGE MANAGEMENT

Q.36. Write a note on change management. *CS : S-13(6M)*

Ans. Change management :

- The repository manages a wide variety of relationships among the data elements stored in it.
- These include relationships between enterprise entities and processes, among the parts of an application design, between design components and the enterprise information architecture, between design elements and deliverables, and so on.
- Some of these relationships are merely associations, and some are dependencies or mandatory relationships. Maintaining these relationships among development objects is called link management.

(5) **Status Reporting :**

- Configuration status reporting (sometimes called status accounting) is a SCM task that answers the following questions :

  (i) What happened?

  (ii) Who did it?

  (iii) When did it happen?

  (iv) What else will be affected?

- The flow of information for configuration status reporting (CSR) is illustrated in Fig.
- Each time a SCI is assigned new or updated identification, a CSR entry is made. Each time a change is approved by the CCS (i.e. an ECO is issued), a CSR entry is made.
- Each time a configuration audit is conducted, the results are reported as part of the CSR task. Output from CSR may be placed in an on-line database or web site so that software developers or maintainers can access change information by keyword category.
- In addition, a CSR report is generated on a regular basis and is intended to keep management and practitioners appraised of important changes.

Q.41. Write short note on version control.

CT : W-09(4M), CS : W-08, 09(4M)

OR Explain version control, change control. *CS : S-09(5M)*

Ans.

(1) Version Control :

- Version Control (also known as Revision Control) is the management of multiple versions of the same unit of information.
- It is most commonly used in engineering and software development to manage ongoing evolution of digital documents like source code, blueprints or electronic models and other critical information that may be worked on by a team of people.
- Version Control is a system or tool that captures the changes to a source code element: file, folder, image or binary.
- Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process.
- A version control system implements or is directly integrated with four major capabilities :

(i) A project database (repository) that stores all relevant configuration objects.

(ii) A version management capability that stores all versions of a configuration object.

(iii) A make facility that enables the software engineer to collect all relevant configuration objects and construct a specific version of the software.

(iv) An issue tracking capability that enables the team to record and track the status of all issues.

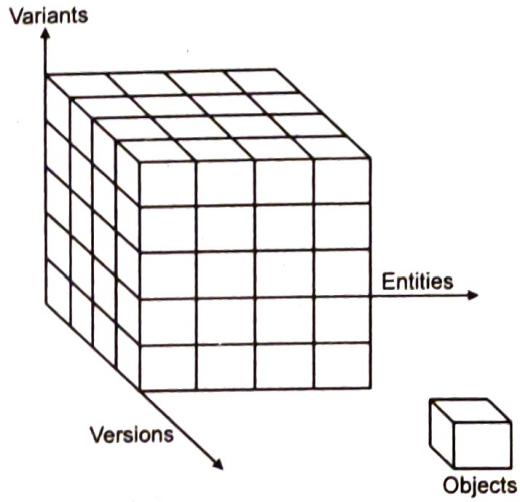


Fig. Object pool representation of components, variants, and versions

- Configuration management allows a user to specify alternative configurations of the software system through the selection of appropriate versions.
- Each node on the graph is an aggregate object, that is a complete version of the software.

(2) Change control :

- Change control is vital. But the forces that make it necessary also make it annoying.
- For a large software engineering project, uncontrolled change rapidly leads to chaos. For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change.

Q.45. Write short note on BPR model.

CT : S-07, W-10(6M), S-09(3M), S-12(5M)

OR Explain BPR model. *CT : S-13(7M), CS : W-13(7M)*

Ans. BPR :

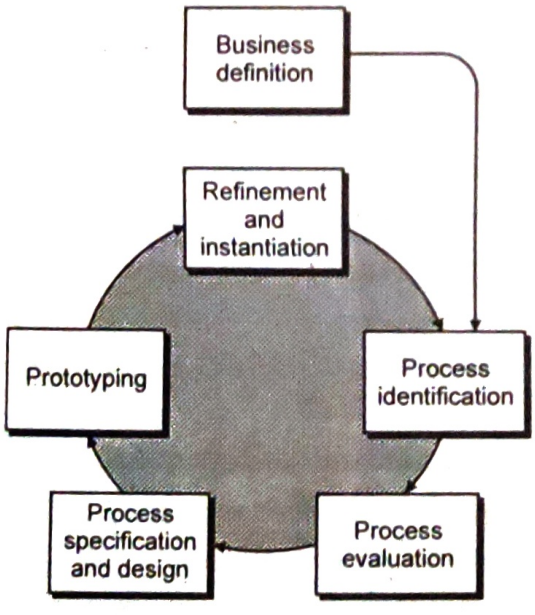
- Like most engineering activities, business process reengineering is iterative.
- Business goals and the processes that achieve them must be adapted to a changing business environment.
- For this reason, there is no start and end to BPR – it is an evolutionary process.
- A model for business process reengineering is depicted in figure.
- The model defines six activities.

(1) Business definition :

- Business goals are identified within the context of four key drivers cost reduction, time reduction, quality improvement and personnel development and empowerment.
- Goals may be defined at the business level or for a specific component of the business.

(2) Process identification :

- Processes that are critical to achieving the goals defined in the business definition are identified.
- They may then be ranked by importance by need for change or in any other way that is appropriate for the reengineering activity.



(3) Process evaluation :

- The existing process is thoroughly analyzed and measured.
- Process tasks are identified the costs and time consumed by process tasks are noted and quality/ performance problems are isolated.

(4) Process specification and design :

- Based on information obtained during the first three BPR activities use cases are prepared for each process that is to be redesigned.
- Within the context of BPR use-cases identify a scenario that delivers some outcome to a customer.
- With the use case as the specification of the process a new set of tasks are designed for the process.

(5) Prototyping :

- A redesigned business process must be prototyped before it is fully integrated into that business.
- This activity "tests" the process so that refinements can be made.

(6) Refinement and instantiation :

Based on feedback from the prototype, the business process is refined and then instantiated within a business system.

SOFTWARE ENGINEERING

Q.46. Explain in detail software reengineering.

CT : S-06, 14, W-12(3M), W-09(6M), S-12(7M)

OR Write short note on software re-engineering process model.

CT : W-07(6M), S-09(5M),CS : S-14(6M)

OR Write a note on software re-engineering.

CT : S-08, 10(3M), W-11(4M)

CS : W-09(5M), S-11(3M), W-12(4M), S-13(7M)

OR What do you mean by reengineering? *CS : S-09(4M)*

OR Provide details of the concept software reengineering. Also highlight the significance of it in present software development scenario. *CT : S-11(8M)*

Ans. Software re-engineering :

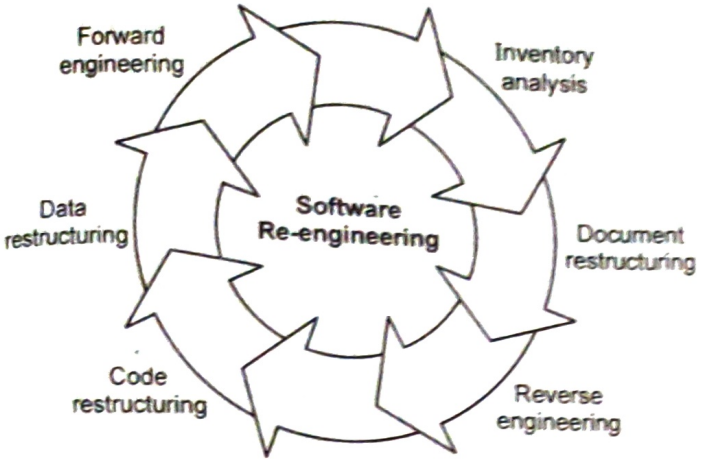


Fig. A software re-engineering process model

- Software re-engineering is the modification of a software system to make it easier to understand and change. Re-engineering often involves software and data restructuring and organization, program simplification and re-documentation.
- Re-engineering is a rebuilding activity and we can better understand the re-engineering of information systems if we consider an analogous activity.
- The re-engineering paradigm shown in the figure is a cyclical model. This means that each of the activities presented as a part of the paradigm may be revisited. The various steps of re-engineering are as follows :

(1) Inventory analysis :

- Sorting active software applications to identify re-engineering candidates. The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.
- By sorting this information according to business criticality, longevity, current maintainability and other locally important criteria, candidates for re-engineering appear.

(2) Document restructuring :

- There are three options for document restructuring as follows:

(i) To live with weak documentation.

(ii) Update poor documents if they are used.

(iii) Fully rewrite the documentation.

- Creating documentation is far too time consuming. It is not possible to re-create documentation for hundreds of computer programs.
- Documentation must be updated. It may not be necessary to fully re-document an application. Rather, those portions of the system that are currently undergoing change are fully documented.
- The system is business critical and must be fully re-documented. Even in this case, an intelligent approach is to spare documentation to an essential minimum.

(3) Reverse engineering :

- Reverse engineering is the process of analysing software with the objective of recovering its design and specification. The program itself is unchanged by the reverse engineering process.
- The software source code is usually available as the input to the reverse engineering process. Sometimes, however, even this has been lost and the reverse engineering must start with the executable code.
- Reverse engineering is not the same thing as re-engineering. The objective of reverse engineering is to derive the design or specification of a system from its source code.
- The process starts with an analysis phase. During this phase, the system is analysed using automated tools to discover its structure.
- 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.

(4) Code 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.
- Source code is analyzed and violations of structured programming practices are noted and repaired.
- The most common type of re-engineering is code restructuring. Some legacy systems have a relatively solid program architecture, but individual modules were coded in a way that makes them difficult to understand, test, and maintain.
- The code within the suspect modules can be restructured. To accomplish this activity, the source code is analysed using a restructuring tool.
- Violations of structured programming constructs are noted and code is then restructured.

- The reverse engineering process should be capable of deriving procedural design representations program and data structure information , data and control flow models and entity relationship models.
- The completeness of a reverse engineering process refers to the level of detail that is provided at an abstraction level. In most cases, the completeness decreases as the abstraction level increases.
- Completeness improves in direct proportion to the amount of analysis performed by the person doing reverse engineering. Interactivity refers to the degree to which the human is "integrated" with automated tools to create an effective reverse engineering process.
- If the directionality of the reverse engineering process is one way, all information extracted from the source code is provided to the software engineer who can then use it during any maintenance activity.

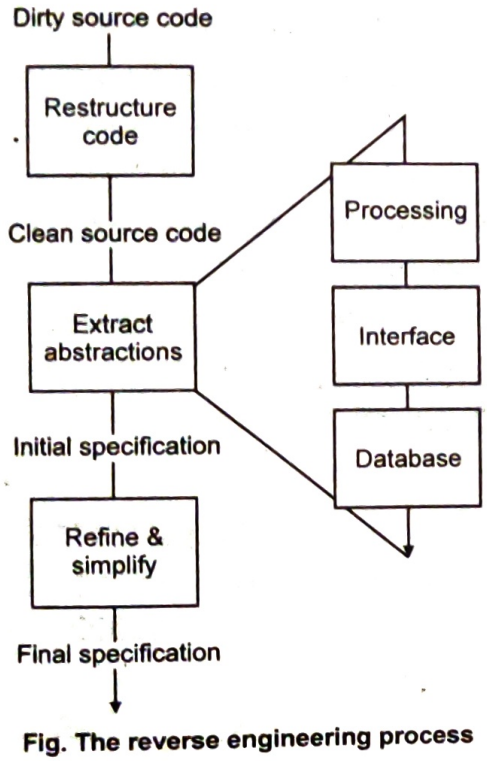


Fig. The reverse engineering process

- The reverse engineering process is shown in the fig.
- Before reverse engineering activities can commence, unstructured source code is restructured, so that it contains only the structured programming constructs.
- The core of reverse engineering is an activity called extract abstractions. The engineer must evaluate the old program and from the (often undocumented) source code, extract a meaningful specification of the processing that is performed, the user interface that is applied and the program data structures or database that is used.

RESTRUCTURING

Q.49. Explain restructuring in details.

OR Write short note on software restructuring. CT-W-07 (3M)

Ans. Restructuring:

- A significant modification made to the debt, operations or structure of a company. This type of corporate action is usually made when there are significant problems in a company, which are causing some form of financial harm and putting the overall business in jeopardy.
- The hope is that through restructuring, a company can eliminate financial harm and improve the business.
- Restructuring does not modify the overall program architecture. It tends to focus on the design details of individual modules and on local data structures defined within modules.
- If the restructuring effort extends beyond module boundaries and encompasses the software architecture, restructuring becomes forward engineering defines a number of benefits that can be achieved when software is restructured:

(i) Programs have higher quality—better documentation, less complexity, and conformance to modern software engineering practices and standards.

(ii) Frustration among software engineers who must work on the program is reduced, thereby improving productivity and making learning easier.

(iii) Effort required to perform maintenance activities is reduced.

(iv) Software is easier to test and debug.

(v) Restructuring occurs when the basic architecture of an application is solid, even though technical internals need work.

Code Restructuring :

- Code restructuring techniques e.g., Warnier's logical simplification techniques model program logic using boolean algebra and then apply a series of transformation rules that yield restructured logic.
- The objective is to take "spaghetti-bowl" code and derive a procedural design that conforms to the structured programming philosophy.
- Restructuring techniques have also been proposed for use with reengineering tools.

(2) **Integrity enforcement :**

- The repository information model also contains rules, or policies, describing valid business rules and other constraints and requirements on information being entered into the repository (directly or via a CASE tool).
- A facility called a trigger may be employed to activate the rules associated with an object whenever it is modified making it possible to check the validity of design models in real time.

(3) **Semantics rich tool interface :**

- The repository information model (meta-model) contains semantics that enable a variety of tools to interpret the meaning of the data stored in the repository.
- For example a data flow diagram created by a CASE tool is stored in the repository in a form based on the information model and independent of any internal representations used by the tool itself.
- Another CASE tool can then interpret the contents of the repository and use the information as needed for its task.
- Thus, the semantics stored in the repository permit data sharing among a variety of tools as opposed to specific tool-to-tool conversions or "bridges".

(4) **Process/project management :**

- A repository contains information not only about the software application itself, but also about the characteristics of each particular project and the organization's general process for software development (phase, tasks and deliverables).
- This opens up possibilities for automated coordination of technical development activity with the project management activity.
- For example updating the status of project tasks could be done automatically or as a by product of using the CASE tools.
- Status updating can be made very easy for developers to perform without having to leave the normal development environment.
- Task assignment and queries can also be handled by e-mail.
- Problem reports, maintenance tasks change authorization and repair status can be coordinated and monitored via tools.

Q.56. Write short note on degree of rigor. *CT : W-07, 11, 12(3M)*

OR What is degree of rigor? Explain different types of degree of rigor. *CT : S-10(6M), W-10(3M), CS : W-08(2M)*

OR How the different project characterization can be done through degree of rigor? *CT : S-13(6M)*

Ans. Degree of rigor:

- The degree of rigor is a function of many project characteristics.
- As an example, small, non-mission critical projects can generally be addressed with somewhat less rigor than large, complex mission critical applications.
- It should be noted, however, that all projects must be conducted in a manner that results in timely, high quality deliverables.
- Four different degrees of rigor are defined for the adaptive process model :

(1) **Casual :**

- All APM framework activities are applied, but only a minimum task set is required.
- In general, umbrella tasks will be minimized and documentation requirements will be reduced.
- All basic principles of software engineering are still applicable.

(2) **Structured :**

- The APM framework will be applied for this project.
- Framework activities and related tasks appropriate to the project type will be applied and umbrella activities necessary to ensure high quality will be applied. SQA, SCM, documentation and measurement tasks will be conducted in a streamlined manner.

(3) **Strict :**

- The APM will be applied for this project with a degree of discipline that will ensure high quality.
- All umbrella activities will be applied and robust documentation will be produced.

(4) **Quick Reaction :**

- The APM will be applied for this project, but because of an emergency situation, only those tasks essential to maintaining good quality will be applied.
- "Back-filling" (e.g., developing a complete set of documentation, conducting additional reviews) will be accomplished after the application/product is delivered to the customer.

Q.57. Explain various steps selecting for task set computation.

CT : S-12(7M)

Ans.

- A task set is a collection of software engineering work tasks, milestones, and deliverables that must be accomplished to complete a particular project.
- The task set to be chosen must provide enough discipline to achieve high software quality.
- Task sets are designed to accommodate different types of projects and different degrees of rigor. Although it is difficult to develop a comprehensive taxonomy of software project types.
- The following steps should be conducted:

(1) Review each of the adaptation criteria and assign the appropriate grades (1 to 5) based on the characteristics of the project.

(2) Review the weighting factors assigned to each of the criteria. The value of a weighting factor ranges from 0.8 to 1.2 and provides an indication of the relative importance of a particular adaptation criterion to the types of software developed within the local environment. If modifications are required to better reflect local circumstances, they should be made.

(3) Multiply the grade, by the weighting factor and by the entry point multiplier for the type of project to be undertaken. The entry point multiplier takes on a value of 0 or 1 and indicates the relevance of the adaptation criterion to the project type. The result of the product grade x weighting factor x entry point multiplier is placed in the produce column for each adaptation criteria individually.

(4) Compute the average of all entries in the Product column and place the result in the space marked task set selector (TSS). This value will be used to help select the task set that is most appropriate for the project.

Q.58. What are the various software components involved in client/server systems ? What are the techniques used to link the various components of client/server architecture ?

CT : S-06(6M)

Ans. The various software components involved in client/server systems:

(1) User interaction/presentation subsystem :

- This subsystem implements all functions that are typically associated with a graphical user interface.

(2) Application subsystem :

- This subsystem implements the requirements defined by the application within the context of the domain in which the application operates.
- For example, a business application might produce a variety of printed reports based on numeric input, calculations, database information, and other considerations.

(3) Database management subsystem.

- This subsystem performs the data manipulation and management required by an application.
- Data manipulation and management may be as simple as the transfer of a record or as complex as the processing of sophisticated SQL transactions.
- Following techniques used to link the various components of client/server architecture.

(i) Distributed presentation :

- In this rudimentary client/server approach, database logic and the application logic remain on the server, typically a mainframe.
- The server also contains the logic for preparing screen information, using software such as CICS. Special PC-based software is used to convert character based screen information transmitted from the server into a GUI presentation on a PC.

(ii) Remote presentation :

- An extension of the distributed presentation approach, primary database and application logic remain on the server and data sent by the server is used by the client to prepare the user presentation.

(iii) Distributed logic :

- The client is assigned all user presentation tasks and the processes

VIVA - VOCE

UNIT - I

Q.1. Define software engineering.

Ans. According to IEEE, software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

Q.2. What are the categories of software?

Ans. The different categories of software are as follows :

(i) System software

(ii) Application software

(iii) Embedded software

(iv) Web Applications

(v) Artificial intelligence software

(vi) Scientific software.

Q.3. What is SDLC?

Ans. A software life cycle deals with various parts and phases from planning to testing and deploying. All these activities are carried out in different ways, as per the needs. Each way is known as a Software Development Lifecycle Model (SDLC).

Q.4. Define process.

Ans. A series of steps involving activities, constraints, and resources that produce an intended output of some kind is known as process.

Q.5. Mention some of the process models appropriate for the software to be engineered.

Ans. Some process models are as follows :

(1) Linear sequential or waterfall model

(2) Prototyping model

(3) RAD model

(4) Incremental model

(5) Spiral model

(6) Winwin spiral model

(7) Component based development model.

Q.6. How does waterfall model works?

Ans. Waterfall model is also called the classic life cycle or linear sequential model, it suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support.

Q.7. What is incremental model?

Ans. The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.

Q.8. What is RAD model?

Ans. The rapid application development (RAD) model is a high speed adaptation of the linear sequential model in which rapid development is achieved by using component based construction.

Q.9. How spiral model works?

Ans. The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall lifecycle model. It also has an emphasis on the use of risk management techniques.

Q.10. What is winwin spiral model?

Ans. Winwin spiral model defines a set of negotiation activities at the beginning of each pass around the spiral. The best negotiations strive for a win-win result.

Q.11. What is formal method model?

Ans. The formal method model encompasses a set of activities that leads to formal mathematical specification of computer software.

Q.12. List the phases of unified process model.

Ans. The phases of unified process modeling are :

(i) Inception

(ii) Elaboration.

(iii) Construction.

(iv) Transition.

(v) Production.