Software Enineering

Chapter 1 : Introduction to Software Engineering and Process Models

Q. 1 Define 'Software Engineering'. **Dec. 15**

Ans. :

Software Engineering

Software Engineering combines two concepts, Software and Engineering.

Software

Software is more than just a program code; it also includes data structures which allow programs to manipulate information, and documentation related to software product which defines the functionality and guidance for the user to use this software.

Engineering

Engineering is an application of knowledge and well defined principles to develop products.

Software Engineering

Software Engineering is the method of applying scientific and technological knowledge, procedures, and rules to design, develop, and maintain the software product.

OR

Applying technological, scientific, and administrative approach to designing, developing, testing and maintaining the software product in order to meet customers' requirements with best quality of product referred as **software engineering**.

Q. 2 Explain in brief the software process framework. **Dec. 15**

Ans. :

Software Process

A software engineering process is the model selected for managing the creation of software from customer initiation to the release of the finished product. The software life cycle follows the sequential flow in order to develop software. The selected process typically involves following methods :

(i) Requirement gathering and analysis

In this phase stakeholders communicate with customer and system users to gather the business requirements like who will use the system? How user will interact with the system? What should be the system input and output? etc.

Depending on these requirements, Requirement Specification Document (SRS) is prepared.

(ii) Design

This phase makes use of output of requirement gathering phase i.e. requirement specification document as an input. Based on requirement specifications software design is prepared. Output of this phase is design specification which specifies hardware and software requirements of system.

Data Flow Diagram, flowcharts etc. are included by design specification document. Design specification acts as an input for implementation phase.

(iii) Implementation phase

Implementation phase of development process decomposes the system work into various smaller parts called **modules**. These modules are assigned to development team members and actual coding is started. This is longest phase of system development. Output of this phase is actual code using specific programming language.

(iv) Testing

Testing phase involves testing of actual code of system against requirements of user in order to ensure that system will satisfy all needs of users. Various testing strategies used are unit testing, system testing, integration testing etc. Output of testing phase is corrected and modified software code.

(v) Deployment

In this phase the system is deployed at users' site for their use. In this phase customer uses the software and give feedback to development team for any changes or modifications in system if any.

(vi) Maintenance

Once the software is deployed actual problems from user's perspective will come up. It is responsibility of development team to solve user's problems time to time. This process is called as maintenance of system. In this phase some additional functionality may need to add in the application as per user's requirement.

Q. 3 Write note on Capability Maturity Model (CMM).

Ans. :

Capability Maturity Model

The Capability Maturity Model (CMM) invented by Software Engineering Institute (SEI) states a growing sequence of levels of a

software development organization. The higher the level, the better the software development procedure so reaching every level is a cost consuming and time consuming procedure. The Capability Maturity Model (CMM) is a mechanism used to create and refine software creation procedure of organization. The CMM is same as ISO 9001, one of the ISO 9000 sequences of standards determined by the International Organization for Standardization (ISO).

The ISO 9000 standards state an efficient quality system for manufacturing and service industries; ISO 9001 applied particularly to software development and maintenance. The major variation among the two systems present in their uses : ISO 9001 states a lowest acceptable quality level for software procedures, while the CMM use a framework for uninterrupted procedure improvement and is more explicit than the ISO standard in defining use of employed to that end.

Following are levels of CMM :

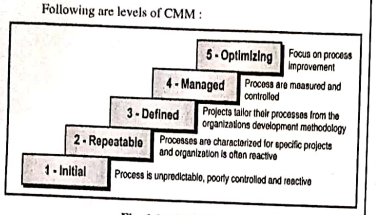


Fig. 1.1 : CMM Levels

1. Level 1 : Initial

This software procedure is stated as inconsistent and occasionally chaotic. Stated procedures and standard practices that present are discarded at the time of crisis. Success of the association mainly depends on efforts taken by each person, capacity, and heroics. The heroes move on to another association taking their knowledge or lessons learnt with them.

2. Level 2 : Repeatable

This level of Software Development association has a fundamental and consistent project management processes to keep the track of cost, schedule, and functionality. The procedure is in region to repeat the latest successes on projects with similar applications. Program management is a most important feature of a level two association.

3. Level 3 : Defined

The software procedure for management and engineering actions are documented, standardized, and incorporated into a standard software procedure for the overall association and each and every project developed by association use an approved, tailored version of the standard software procedure of association for creating, testing and maintaining the software product.

4. Level 4 : Managed

Management can successfully control the efforts required for software development through precise measurements. At this level, association set a quantitative quality aim for both software procedure and software maintenance. At this level, the performance of procedures is controlled through statistical and other quantitative mechanisms, and is quantitatively predictable.

5. Level 5 : Optimizing

The important characteristic of this level is concentrating on repeatedly improving procedure performance with the help of both increasing and innovative technological corrections. At this level, modifications to the procedures are done to increase the procedure performance and at the same time maintaining statistical probability to get the recognized quantitative procedure-improvement aims.

Q. 4 Explain the waterfall model.

Ans. :

Waterfall Model

Waterfall model is the first approach used in software development process. It is also called as classical life cycle model or linear sequential model. In waterfall model any phase of development process begins only if previous phase is completed.

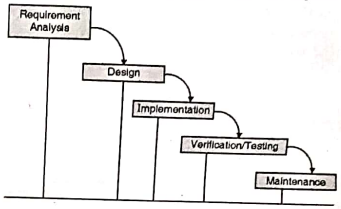


Fig. 1.2 : Waterfall Model

1. Requirement Analysis

In this phase all business requirements of system are gathered and analyzed by communication between stakeholders and customer or user.

At the end of this phase Requirement Specification Document (SRS) is created.

2. Design

Based on requirement, specification document design of system is created called software architecture. It is blue print of system representing system's internal structure and behavior.

3. Implementation

In implementation phase actual coding is constructed for software architecture using hardware and software requirements of system. It is responsibility of developer.

4. Verification / Testing

Here coding or job done by developer is verified against requirements of user in order to ensure that software will satisfy all business requirements of user. After the successful verification software is deployed at user's site for their use.

5. Maintenance

While using software if user faces some problems, then those problems must be solved time to time by development team. This task comes under maintenance of software. Maintenance also includes adding new functionalities in software as per user requirements.

Advantages of waterfall model

(i) It is very simple to understand and easy to use.

(ii) Phases of waterfall model do not overlap with each other.

(iii) It is useful for small projects in which requirements are clear initially.

(iv) Since development is linear it is easy to manage development process.

Disadvantages of waterfall model

(i) It is not useful for large projects.

(ii) Not suitable for projects in which requirements are not clear initially.

(iii) System or product is available only at the end of development process.

(iv) It is very difficult to modify system requirements in the middle of development process.

Practical situations in which Waterfall model can be used

1. This model is used only when the requirements are very well known, clear and fixed.
2. Product definition is stable.
3. Technology is understood.
4. There are no ambiguous requirements.
5. Ample resources with required expertise are available freely.
6. The project is short.

Q. 5 Discuss incremental model for software development with merits and demerits. Dec.-16

Ans. :

Incremental Process Models

The incremental model applies the waterfall model incrementally. The series of releases is referred to as "increments", with each increment providing more functionality to the customers. After the first increment, a core product is delivered, which can already be used by the customer. Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly. This process continues with increments being delivered until the complete product is delivered. The incremental philosophy is also used in the agile process model.

Following tasks are common to all the models :

1. **Communication :** Helps to understand the objective.
2. **Planning :** Required as many people (software teams) work on the same project but different functions at same time.
3. **Modelling :** Involves business modelling, data modelling, and process modelling.
4. **Construction :** This involves the reuse of software components and automatic code.
5. **Deployment :** Integration of all the increments.

Characteristics of Incremental Model

1. System is broken down into many mini development projects.
2. Partial systems are built to produce the final system.
3. First tackled highest priority requirements.
4. The requirement of a portion is frozen once the incremented portion is developed.

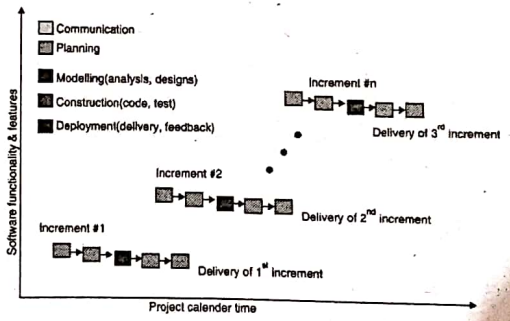


Fig. 1.3 : Incremental Model

Advantages of Incremental Model

1. After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
2. It is generally easier to test and debug than other methods of software development because relatively smaller changes are

made during each iteration. This allows for more targeted and rigorous testing of each element within the overall product.

3. Customer can respond to features and review the product for any needed or useful changes.
4. Initial product delivery is faster and costs less.

Disadvantages of Incremental Model

1. Resulting cost may exceed the cost of the organization.
2. As additional functionality is added to the product, problems may arise related to system architecture which were not evident in earlier prototypes.

Q. 6 With a neat diagram explain the Spiral model of software development. **Dec. 17**

Ans. : Spiral Model

Spiral model is a combination of iterative model and waterfall model. Spiral model has four phases of development each of these phases is called as spiral.

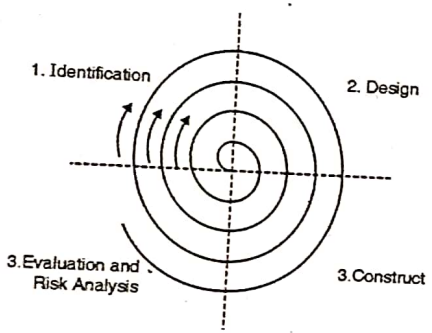


Fig. 1.4 : Spiral Model

(i) Identification

This phase identifies all business requirements of system at the beginning. It involves clear understanding of requirements by communication between stakeholders and customer. In the subsequent iterations all subsystem requirements and unit requirements are identified.

(ii) Design

In first iteration, design phase develops conceptual design of system based on initially gathered requirements. In further spirals or iterations, it develops logical design, physical design, architectural design and final design of system.

(iii) Construct

Initially construct phase develops a code for conceptual design to get user feedback. In next subsequent spirals, detailed working model of software is constructed with increment number and are delivered to customer for feedback.

(iv) Evaluation and risk analysis

In this phase management risks like cost overrun are identified and monitored, technical feasibility of system is also done. At end of each spiral customer evaluates software for potential risks in system and provides feedback. Spiral model can be used for projects where budget and risk evaluation are important factors. If customers are not sure about their requirements then spiral model is useful than waterfall model.

Advantages of spiral model

(i) It is more flexible to changing requirements.
(ii) Requirements are achieved more accurately.
(iii) User can see the system from 1st iteration to end of development.
(iv) Risk management is easier.

Disadvantages of spiral model

(i) It is difficult to manage development process.
(ii) Not useful for small projects development.
(iii) Spiral can run indefinitely.
(iv) It requires excessive documentation work as documentation is prepared for each iteration.

Q. 7 Discuss prototyping model for software development with merits and demerits. **Dec. 16, Dec. 17**

Ans. : Prototyping Model

Prototyping model refers to developing software application prototypes (early approximation / version) which displays the behaviour of product under development but may not actually contain the exact logic of the original application. Prototyping allows user to evaluate developers' proposal and try it before actual implementation. Prototyping model is widely used popular software development model as it helps to understand user requirements in early stage of development process.

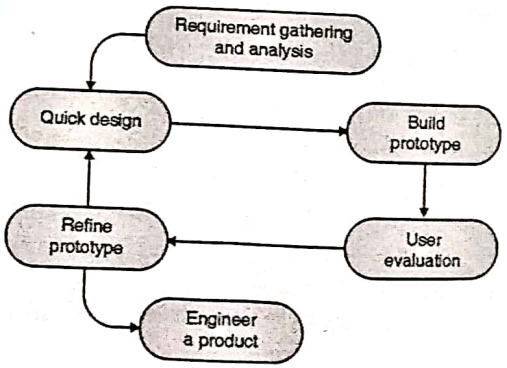


Fig. 1.5 : Prototyping Model

Phases of prototyping model are as follow :

(i) Requirement gathering and analysis

Building of prototype begins with requirement gathering and analysis. In this phase various users of system are interviewed in order to know their system requirements or expectations from system. Requirement specification report is generated as an output of this phase.

(ii) Quick Design

After gathering and analysis of user requirements quick design (prototype design) of system is developed. Quick design is not detailed design of system; it contains only necessary characteristics of the system which gives basis idea of system to the end users.

(iii) Build prototype

Based on feedback received from user about quick design of system, the first prototype of system is created. Prototype is working model of system under development.

(iv) Use Evaluation

Once prototype is built, proposed system is presented to end user of system for its evaluation. User determines strengths and weaknesses of prototype i.e. what needs to be added or what is to be removed. All the comments and suggestions given by user in feedback are passed to developer.

(v) Refining prototype

Depending on comments and suggestions came from user, developers refine previous prototype to form new prototype of system. New prototype is again evaluated just like previous prototype. The process of evaluation and refining prototype continues until all user requirements are met by prototype.

(vi) Engineer a product

Once evaluation and refining of prototype completes i.e. when user accepts final prototype of system the final system is evaluated thoroughly and deployed at user's site. Deployment of engineered product is further followed by regular maintenance of system.

Advantages of prototyping model

(i) It enables early evaluation of system by providing working model to end users at early stage of development.

(ii) Refining of prototype results in better implementation of system requirements.

(iii) Communication between developer and user reduces ambiguity.

(iv) More involvement of user in development process results in meeting user's requirement at greater extent.

Disadvantages of prototyping model

(i) Refining of prototype continues until user is completely satisfied, thus it is time consuming and expensive process.

(ii) More involvement of user in development process is not accepted by developer always.

(iii) Refining of prototype again and again may disturb the working of development team.

(iv) Practically prototyping model results in increasing the complexity of system as scope of system extends beyond original plan.

Q. 8 Write short note on Agile Process. **May 15, Dec. 15, May 16, Dec. 16, May 17, Dec. 17, May 18**

Ans. :

Agile Process

Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customers (end users). It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change. The term Agile was popularized, in this context, by the Manifesto for Agile Software Development.

The values and principles adopted in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban. There is significant subjective evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations.

Agile software development values

Based on their combined experience of developing software and helping others do that, the seventeen signatories to the manifesto proclaimed that they value :

1. Individuals and Interactions over processes and tools.
2. Working Software over comprehensive documentation.
3. Customer Collaboration over contract negotiation.
4. Responding to Change over following a plan.

As per the view of Scott Ambler :

1. Tools and processes are important, but it is more important to have competent people working together effectively.
2. Good documentation is useful in helping people to understand how the software is built and how to use it, but the main point of development is to create software, not documentation.
3. A contract is important but is no substitute for working closely with customers to discover what they need.

4. A project plan is important, but it must not be too rigid to accommodate changes in technology or the environment, stakeholders' priorities, and people's understanding of the problem and its solution.

Agility Principles

The Manifesto for Agile Software Development is based on twelve principles :

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Working software is delivered frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

Advantages of Agile Process

There are number of advantages of Agile Process :

1. Stakeholder Engagement,
2. Transparency,
3. Early and Predictable Delivery,
4. Predictable Costs and Schedule,
5. Allows for Change,
6. Focuses on Business Value,
7. Focuses on Users,
8. Improves Quality.

Q. 9 Explain Extreme Programming (XP) with suitable diagram. **Dec. 15, May 17**

Ans. :

Extreme Programming (XP)

Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.

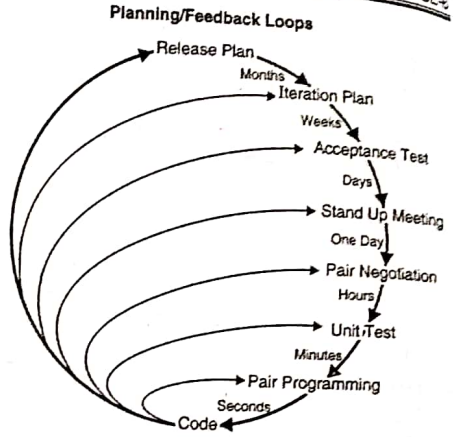


Fig. 1.6 : Extreme Programming

As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted. Other elements of extreme programming include : Programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, code simplicity and clarity, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers.

The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels. As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed continuously, i.e. the practice of pair programming.

XP Values

Extreme programming initially recognized four values in 1999 : communication, simplicity, feedback, and courage. A new value, respect, was added in the second edition of Extreme Programming Explained.

1. Communication

The goal is to give all developers a shared view of the system which matches the view held by the users of the system. To this end, extreme programming favors simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.

2. Simplicity

Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later. The difference

between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month.

This is sometimes summed up as the "You aren't gonna need it" (YAGNI) approach. Related to the "communication" value, simplicity in design and coding should improve the quality of communication. A simple design with very simple code could be easily understood by most programmers in the team.

3. Feedback

Within extreme programming, feedback relates to different dimensions of the system development :

1. Feedback from the system
2. Feedback from the customer
3. Feedback from the team

Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will break.

4. Courage

This is an effort to avoid getting delayed in design and requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with refactoring their code when necessary. This means reviewing the existing system and modifying it so that future changes can be implemented more easily. Also, courage means persistence: a programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, but only if they are persistent.

5. Respect

The respect value includes respect for others as well as self-respect. Programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers. Members respect their own work by always striving for high quality and seeking for the best design for the solution at hand through refactoring.

Q. 10 Write short note on SCRUM. **Dec. 17**

Ans. :

Scrum

Scrum is an agile framework for managing knowledge work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within time boxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute stand-up meetings, called daily scrums.

Scrum principles are mostly based on the agile manifesto and help to guide activities regarding development inside a process which includes framework activities such as requirements, analysis, design, evolution, and delivery.

In all the framework activities, work tasks happen within a process pattern called a sprint.

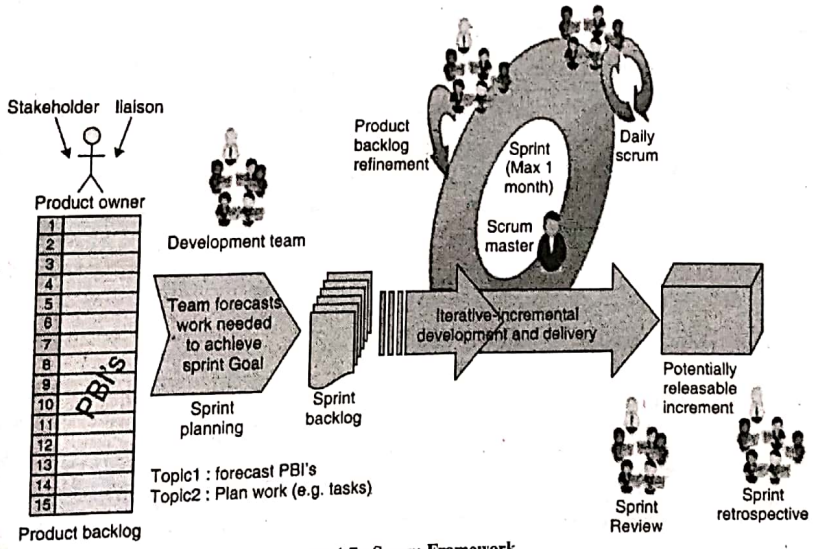


Fig. 1.7 : Scrum Framework

The work implemented in a sprint is tailored to the problem at hand and is defined and usually tailored in real time by the respective Scrum team.

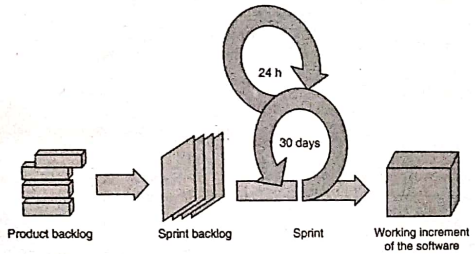


Fig. 1.8 : The scrum process

Scrum mostly focus on the use of a bunch of software process patterns which are proved effective for projects with restrictive timelines, changing requirements, and business criticality. All such process patterns defines several sets of development actions :

1. **Backlog** - A prioritized list regarding project needs or features which offer business value for the customer. It is possible to add items to the backlog at any time. The respective product manager assesses the backlog and changes the priorities as per necessity.
2. **Sprints** - Includes work units which are necessary to gain a requirement defined in the backlog which should be incorporated in predefined time-box (usually thirty days).

Modifications (such as backlog work items) are not involved during the sprint. Therefore, team members are allowed by the sprint to work in a short-term, but stable environment.

Scrum meetings - are short (usually fifteen minutes) meetings conducted on daily basis by the Scrum team. Three most important key questions which are asked and must be answered by all team members are :

1. What did you do since the last team meeting ?
2. What obstacles are you encountering ?
3. What do you plan to accomplish by the next team meeting ?

A team leader, who is known as a Scrum master, handles the meeting and evaluates the responses from each person. The important use of Scrum meeting to the team is that it can uncover critical issues as early as possible. One more benefit of these daily meetings is that they lead to "knowledge socialization" and thus encourage a self-organizing team structure.

3. **Demos** - implement the software increment at the client side (customer) so that the newly developed functionality can be demonstrated as well as evaluated by the customer.

It is also possible that the demo may not able to contain all planned functionality, but instead such functions which can be delivered within the established time-box.

Q. 11 Differentiate between Scrum and Kanban Model.

Ans. :

Comparison between Scrum and Kanban

Parameter	SCRUM	KANBAN
Tempo	Regular fixed length sprints (i.e. 2 weeks)	Continuous flow
Release methodology	At the end of each sprint if approved by the product owner.	Continuous delivery or at the team's discretion.

Roles	Product owner, scrum master, development team	No existing roles. Some teams enlist the help of an agile coach.
Key metrics	Velocity	Cycle time.
Change philosophy	Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learning's around estimation.	Change can happen at any time.

Chapter 2 : Requirements Analysis and Modelling

Q. 1 What do you mean by requirement ? Enlist Characteristics of requirements. **Dec. 16**

Ans. :

Requirement

The information which describes the user's expectation about the system performance is called as requirement. The requirement must be clear and unambiguous. Some requirement definition has to face problem of lack of clarity.

Characteristics of requirements

There are various characteristic of requirements. They are as follows :

1. **Requirements should be unambiguous**

   Ambiguous means the single word or statement has more than one meaning. If requirements contain ambiguity then it is difficult to fulfill the requirements correctly. Therefore the requirements should be unambiguous means non confusing.

2. **Requirements should be testable (verifiable)**

   The requirements should be testable means tester should be able to easily test the requirement whether they are implemented successfully or not. For easy test, the requirement should be clear and unambiguous.

3. **Requirements should be clear (concise, simple, precise)**

   The requirements should be clear. They should not contain any unnecessary information. If there is any unnecessary information then it becomes difficult to achieve the appropriate fulfillment of the requirement.

4. **Requirements should be understandable**

   The requirements should be understandable means when anyone read it then it should be easily understood by that person. To make it understandable proper conventions are used. It should be grammatically correct.

5. **Requirements should be feasible (realistic, possible)**

   The requirement should be completed within the given time and budget. Specified requirement said to be feasible if it can be implemented using existing technology, with estimated budget and time.

6. **Requirements should be Consistent**

   Consistency is an important feature of requirements. It means that all inputs must be processed similarly. It should not happen that processes produce different outputs for same inputs coming from different sources.

Q. 2 Mention any four requirement Elicitation Methods. **Dec. 17**

Ans. :

Requirement elicitation methods

In requirements engineering, requirements elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. The practice is also sometimes referred to as "requirement gathering". The term elicitation is used in books and research to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering.

Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do OR NOT do (for Safety and Reliability). Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping. Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process. Requirements elicitation is a part of the requirements engineering process, usually followed by analysis and specification of the requirements.

There are different ways to identify customer requirement :

(1) Interviews

Interviews are important way for gathering requirements. Organization may take various kinds of interviews such as :

1. Structured or closed interviews in which information to be collected from customer is decided in advance.
2. Non-structured or open interviews in which details needs to be collected from customer are decided in advance.

3. Oral interviews.
4. Written interviews.
5. Face-to-face interviews which are held among two people across the table.
6. Groups of persons are participating in group interviews. They support to cover any missing requirement as number of people participates in this process.

(2) Surveys

Organization may perform surveys of different stakeholders by questioning them about their requirements and expectation from the proposed software system. The advantage of this technique is that, collecting requirements is economically beneficial because it collects requirements from a large number of persons at same time.

Surveys are less effective method of data discovery. Frequently, a survey's main finding is that other questions should have been asked instead. Surveys are most useful for capturing clear factual information.

(3) Questionnaires

Questionnaires are a document way which contains in-built set of objectives that is based on questions and their options. This document is given to all stakeholders to give answer of those questions which are gathered and compiled. Output of this mechanism is, if an answer for some question is not given in the questionnaire, the issue may be remaining unattended. The advantage of this technique is that, collecting requirements economically is beneficial because it collects requirements from huge amount of persons at same time. Questionnaires are less effective method of data discovery.

(4) Task analysis

In this technique, team of software developers and engineers may identify the functional specifications for which the new system is needed to develop. If the customer has some software to do the particular operations then it is analyzed by this team to find out requirements for proposed system.

(5) Domain Analysis

Each software put into some domain category. The experienced persons in that domain can be a great support to study general and specific requirements.

(6) Brainstorming

Brainstorming is an informal debate held between different stakeholders and all their suggestions are documented for further requirement analysis.

(7) Prototyping

Prototyping is a technique in which, we create user interface without including detail functionality for user to interpret features of desired software product. It supports providing details about requirements. If the client does not know its own requirements, in such case the developer develop a prototype based on requirements provided at initial stage.

The prototype is displayed to the client and the feedback is collected from client. The client feedback is used as an input for requirement gathering.

(8) Observation

Team of experienced persons visits the organization or workplace of client. They observe existing system's work. They observe the flow of control at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Q. 3 Explain Functional and Non-Functional Requirements with types. **Dec. 16**

Ans. :

Functional Requirements

The Functional requirements specification create document regarding the operations as well as activities which a system must be able to carry out. In software engineering, a functional requirement is used to define a function regarding a system or its component, in which a function is described as a specification of behavior between outputs and inputs. In the functional requirements there may be elements such as calculations, technical details, data manipulation and processing, and other specific functionality which define what a system is supposed to implement.

As defined in requirements engineering, functional requirements is used to mention specific results of a system. This should be contrasted with non-functional requirements that mention overall characteristics like as cost and reliability. Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.

Functional Requirements should include

1. Descriptions of data to be entered into the system.
2. Descriptions of operations performed by each screen.
3. Descriptions of work-flows performed by the system.
4. Descriptions of system reports or other outputs.
5. Persons who can enter the data into the system.
6. Way the system meets applicable regulatory requirements.

Examples of Functional Requirements

Functional requirements must contain functions performed by specific screens, outlines of work-flows performed by the system, and other business or compliance requirements the system must meet.

Interface requirements

1. Field 1 accepts numeric data entry.
2. Field 2 only accepts dates before the current date.
3. Screen 1 can print on-screen data to the printer.

Business Requirements

1. Data must be entered before a request can be approved.
2. Clicking the Approve button moves the request to the Approval Workflow.
3. All personnel using the system will be trained.

Regulatory/Compliance Requirements

1. The database will have a functional audit trail.
2. The system will limit access to authorized users.
3. The spreadsheet can secure data with electronic signatures.

Security Requirements

1. Members of the Data Entry group can enter requests but cannot approve or delete requests.
2. Members of the Managers group can enter or approve a request but cannot delete requests.
3. Members of the Administrators group cannot enter or approve requests but can delete requests.

Non Functional Requirements (NFR)

These are basically the quality constraints that the system must satisfy according to the project contract.

The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like :

1.	Portability	2.	Security
3.	Maintainability	4.	Reliability
5.	Scalability	6.	Performance
7.	Reusability	8.	Flexibility

NFRs are classified into following types :

1. Interface constraints
2. Performance constraints : response time, security, storage space, etc.
3. Operating constraints
4. Life cycle constraints : maintainability, portability, etc.
5. Economic constraints

The process of specifying non-functional requirements needs the knowledge of the functionality of the system, and also the knowledge of the context within which the system will operate.

Types of Non-Functional Requirements

1. **Product requirements :** Requirements which mention that the delivered product should have behavior in a specific way, e.g. execution speed, reliability etc.
2. **Organizational requirements :** Requirements that are a result of organizational policies as well as procedures, such as process standards which has been used, implementation necessities etc.
3. **External requirements :** Requirements which are generated from the factors that are external to the system and its development process, e.g. interoperability requirements, legislative requirement.

Q. 4 Explain SRS document ? **May 15**

Ans. : Software Requirement Specification (SRS)

SRS stands for Software/System Requirement Specification. SRS is a special kind of document which contains user requirements for a system which states properties and constraints that must be satisfied by a software system. IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces. The outcome of the requirements gathering and analysis phase of the SDLC is **Software Requirements Specification (SRS).** SRS is also called as **requirements document.** SRS is base for software engineering actions and is generated when all requirements of software project are gathered and analyzed. It is generally signed at the end of requirements engineering phase.

Following are 6 requirements stated by Heninger, which SRS document should follow :

1. SRS document should specify external system behaviour.
2. SRS document should specify implementation constraints.
3. It should be easily changeable if any changes occur.
4. It should act as reference tool for maintaining the system.
5. SRS document record forethought about the lifecycle of the system i.e. predicts changes.
6. It must include acceptable response to undesired events.

Advantages of SRS

(1) SRS contains the base for agreement among the client and the company who develop the product as per expectations.

(2) A SRS gives a base for verification of the completely developed product.

(3) Using high-quality SRS, we can develop high-quality software product.

(4) A high-quality SRS decreases the cost of development.

Characteristics of SRS

A good software requirement specification must satisfy following characteristics :

1. Correct

SRS is correct when all requirements of user are described in the requirement document. The listed requirements must be matched with desired system. This depict that every requirement is analyzed to give assurance that it (SRS) contains user requirements. Remember that there is no specific tool or process to ensure the correctness of SRS. Correctness gives assurance that all stated requirements are worked as expected.

2. Unambiguous

SRS does not contain any confusion when each specified requirement has single interpretation. This characteristic state that every requirement is individually interpreted. In situation, where one term has number of meanings, then its meaning must be specified in the SRS so that it will be non-confusing and simple to understand.

3. Complete

SRS is complete when the requirements undoubtedly specified that which work the software product is required to perform. This contains each and every requirement associated to performance, design and functionality.

4. Ranked for importance/stability

Each and every requirement has not same importance, so every requirement is recognized to make differentiation between requirements. For this purpose, it is needed to undoubtedly recognize every requirement. Stability refers to the probability of further modifications in the requirement.

5. Modifiable

The requirements given by user are changeable, so requirement document must be generated in such a way that those modifications can be included easily in SRS by preserving consistently the structure and style of the SRS.

6. Traceable

SRS is observable when the source of every requirement is unambiguous and facilitates the description of every requirement in future. Forward tracing and backward tracing methods are used for this purpose. Forward tracing state that every requirement must be referencing to design and code of components. Backward tracing state every requirement explicitly addressing its source.

7. Verifiable

SRS is testable when the stated requirements can be tested with a cost-effective procedure to verify whether the final software fulfills those requirements. The requirements are tested through reviews. Clear requirement is needed for verifiability.

Q. 5 Build an SRS document for online student feedback system. **May 08**

Ans. :

SRS Document for Online Student Feedback System

Table of Contents

1. Introduction

1.1 Purpose

This document gives detailed functional and non-functional requirements for online student feedback system. The purpose of this document is that the requirements mentioned in it should be utilized by software developers to implement the system.

1.2 Scope

This system allows the students to provide quick feedback which is provided by collage staff. The feedback report is generated and which is checked by HOD's. He can view grade and grade obtained to the lecturers.

1.3 Overview

This system provides an easy solution to collage staff and students for maintaining feedback related to collage staff and infrastructure, facility.

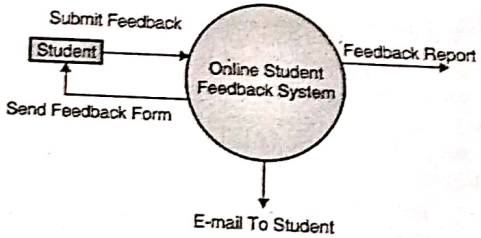


Fig. 2.1 : Online student feedback system

2. General Description

This online student feedback system replaces the traditional, manual feedback system by which lots of paper work will be reduced. The teachers are able to provide feedback regarding facility and students are able to provide feedback easily. This is primary feature of this system.

Another feature is that feedback form can be provided to student and staff by emailing for filling it.

2.1 User Manual

The system should provide Help option in which how to operate system should be explained. Also hard copy of this document should be given to user in booklet form.

3. Functional Requirement

3.1 Description

For identity of staff, system should display staff photograph along with their names for that corresponding subject and skills.

Statistical report accordingly subject or skill should display individual's report whenever required.

4. Interface Requirement

4.1 GUI

Student

Student can give the feedback about the lecturers on the scale of ten. Student can give feedback about the lecturers based on interaction of lecturer in the class rooms with students and facility provided by collage like infrastructure etc.

Staff

The feedback given by students can be viewed by the staff and improve their performance in teaching and other aspects.

Head of Department

The feedback report can be checked by HOD. He can view overall grades and view the grades obtained by the lecturers and give report to principal and he can give counseling to the collage staff.

Principal

Finally, report was referred by principal who can give suggestions to the lecturers to improve their teaching.

4.2 Hardware Interface

The system should be embedded in all desktops.

4.3 Software Interface

(i) Online Student Feedback System

(ii) The feedback database transmitted to database server.

(iii) Report generator.

5. Performance Requirements

The system should work concurrently on multiple processors between the collage hours.

The report should be generated immediately within one hour.

6. Design Constraints

The system should be designed within 6 months.

7. Other non-functional attributes

7.1 Security

The student and staff should be provided password to log on to the system.

7.2 Reliability

Reliability should be guaranteed due to wired connectivity.

7.3 Availability

The system should be available during college hours.

7.4 Maintainability

There should be facility to add and delete feedback form for different purpose.

7.5 Reusability

The same system will be used in each semester.

8. Operational Scenarios

There should be student database and teacher database. The student database should contain name and feedback information.

The teacher database should contain name, subject, skills, and other details.

9. Preliminary Schedule

The system should be designed within 6 months.

Q. 6 Develop a software requirement specification (SRS) for developing a software for Hospital Management System.

Create an SRS that contains following:

1. Objective and scope

2. Product Perspective

3. Functional Requirement

4. Non Functional Requirement **May 17, May 18**

Ans. : SRS for Hospital Management System

Title

System Requirement Specification Document for Hospital Management System.

Objective

To get with preparing requirement document, which will be used to capture and document all the requirements at the start of project. In the assignment we mainly focus on functional requirements.

1. Introduction

1.1 Purpose

The main purpose of our system is to make hospital task easy and is to develop software that replaces the manual hospital system into automated hospital management system. This document serves as the unambiguous guide for the developers of this software system.

1.2 Document Conventions

HMS - Hospital Management System

GUI - Graphical User Interface

PHID - Patient Hospital Identification Number

1.3 Scope of the Project

The purpose of this specification is to document requirements for a system to manage the hospital.

The specification identifies what such a system is required to do.

The Hospital Management System will manage a waiting list of patients requiring different treatments.

The availability of beds will be determined and if beds are available the next appropriate patient on the list will be notified.

Nurses will be allocated to wards depending on ward sizes, what type of nursing is needed, operating schedules, etc.

The current manual method of managing patients, nurses, and beds is time consuming and error prone. It is also difficult to manage the large paper flow involved in this process.

The Hospital Management System will allow hospital administrative staff to access relevant information efficiently and effectively.

The goal of HMS is to manage nurses, patients, beds, and patients' medical information in an cost-effective manner.

All of these sub-systems (managing nurses, beds, patient medical information) need to be designed and implemented so that HMS can run effectively.

2. Overall Description

2.1 Product Perspective

The HMS is designed to help the hospital administrator to handle patient, nurse and bed information. The current design goal is to build an internal system to achieve the functionality outlined in this specification.

2.2 Product Functions

The HMS will allow the user to manage information about patients, nurses, and beds. Patient management will include the checking-in and checking-out of patients to and from the hospital.

The HMS will also support the automatic backup and protection of data.

2.3 Operating Environment

Following are the requirements for running the software successfully :

Processor – Pentium III or Higher.

Ram – 512 MB or Higher.

Disk Space – 10 GB or Higher.

OS – Windows XP or Above.

2.4 Design and Implementation Constraint

GUI only in English.

Login and password is used for identification of user and there is no facility for guest.

2.5 Assumption and Dependencies

It is assumed that one hundred compatible computers will be available before the system is installed and tested.

It is assumed that Hospital will have enough trained staff to take care of the system.

3. External Interface Requirements

3.1 User Interface

Input from the user will be via keyboard and mouse. The user will navigate through the software by clicking on icons and links. The icons will give appropriate responses to the given input.

3.2 Hardware Interface

These are the minimum hardware interfaces :

Processor : Pentium III or Higher.

Ram : 512 MB or Higher.

Disk Space : 10 GB or Higher.

3.3 Software Interface

These are the minimum software interfaces :

Technologies : C# .Net 2.0

Database : SQL server (standard edition).

Operating system : Windows XP or above.

4. System Features

4.1 System Features

Work Scheduling : Assigning nurses to doctors and doctors to patients.

Admissions : Admitting patients, assigning the patients to appropriate wards.

Patient Care : Monitoring patients while they are in the hospital.

Surgery Management : Planning and organizing the work that surgeons and nurses perform in the operating rooms.

Ward Management : Planning and coordinating the management of wards and rooms.

Waiting list : Monitoring to see if there are any patients waiting for available beds, assigning them to doctors and beds once these become available.

5. Other Non-functional Requirements

5.1 Performance Requirements

The performance of our software is at its best when the following regularly are done :

1. Password Management
2. Regular Database Archiving
3. Virus Protection

5.2 Safety Requirements

Humans are error-prone, but the negative effects of common errors should be limited. e.g. users should realize that a given command will delete data, and be asked to confirm their intent or have the option to undo.

5.3 Security Requirements

Each member is required to enter an individual Username and password when accessing the software. Administrators have the option of increasing the level of password security their members use.

The data in the database is secured through multiple layers of Protection.

One of those security layers involves member passwords. For maximum Security of your software, each member must protect their password.

5.4 Software Quality Attributes

The Quality of the system is maintained in such a way that it can be very user-friendly. The software quality attributes are assumed as follows :

1. Accurate and hence reliable.
2. Secured.
3. Fast Speed.
4. Compatibility.

Q. 7 Write note on class based model.

Ans. : Class-based Model

The generic process described starts with guidelines for identifying potential analysis classes, suggestions for defining attributes and operations for those classes, and a discussion of the Class-Responsibility-Collaborator (CRC) model.

The CRC card is used as the basis for developing a network of objects that comprise the object-relationship model.

Identifying Analysis Classes

The problem statement should be examined to identify analysis classes. For isolation of potential classes a "grammatical parse" can be used. Identify each and every attribute of all the classes. Identify operations which lead to manipulation of the attributes.

There are number of ways by which *Analysis Classes* manifest themselves :

1. **External entities** such as several systems, devices and people that generate or use information to be referred by a computer-based system.
2. **Things** such as reports, displays, letters and signals which are considered as part of the information domain for the respective problem.
3. **Occurrences** or **events** such as a property transfer or the finishing point of a series regarding robot movements that take place within the context of system operation.
4. **Roles** such as manager, engineer and salesperson played by individuals who need to interact with the system.
5. **Organizational units** such as division, group, and team which are basically relevant to an application.
6. **Places** such as manufacturing floor or loading dock that establish the context regarding the respective problem and the overall function of the system.
7. **Structures** such as sensors, four-wheeled vehicles, or computers which define a class of objects or related classes of objects.

One can extract the nouns by performing a "grammatical parse" on a processing narrative for a problem. After the process of identification of the nouns, several potential classes are proposed in a list. The list gets increased until all of the nouns present in the processing narratives have been considered. Each and every entry in the list is considered as a potential object.

To determine whether a potential class can become an analysis class following characteristics are measured :

1. **Retained Information :** The potential class is considered as helpful in the process of analysis only if information regarding it, must be remembered for the functioning of so the system.
2. **Needed Services :** The potential class should posses a set of identifiable operations which have ability to change the value of its attributes in some way.
3. **Multiple attributes :** During R.A., the centre of attention must be "major" information; it is also possible that a class having single attribute may be useful in design, but it's better to represent it as an attribute of another class during the analysis activity.
4. **Common attributes :** It is possible to define a set of attributes for the potential class, and these defined attributes can be applied to all instances of the class.
5. **Common operations :** It is possible to define a set of operations for the potential class, and these defined operations can be applied to all instances of the class.
6. **Essential Requirements :** External entities which appear in the problem space and generate or use information necessary to the operation of any solution for the respective system will nearly always be defined as classes in the requirement model.

Specifying Attributes

Attributes are set of data objects which completely define the class within the context of the problem space. For the purpose of developing a meaningful set of attributes for an analysis class, a programmer can revise a use-case and choose such things which "reasonably" belong to the class.

A vital aspect that should be concerned for each class : what data items fully define the class in the context of the problem at hand.

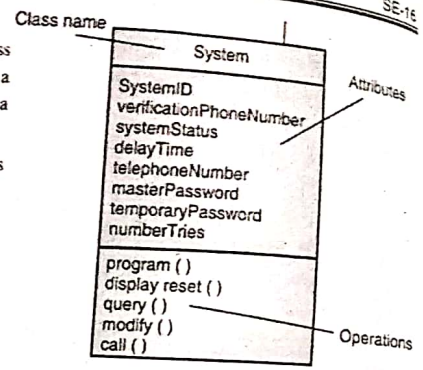


Fig. 2.2 : Specifying Attributes

Defining Operations

Operations which define the behavior of an object are divided into 4 broad categories :

1. Operations which manipulate data (adding, deleting, selecting, reformatting.)
2. Operations which carry a computation.
3. Operations which inquire regarding the state of an object.
4. Operations which monitor an object for the occurrence of a controlling event.

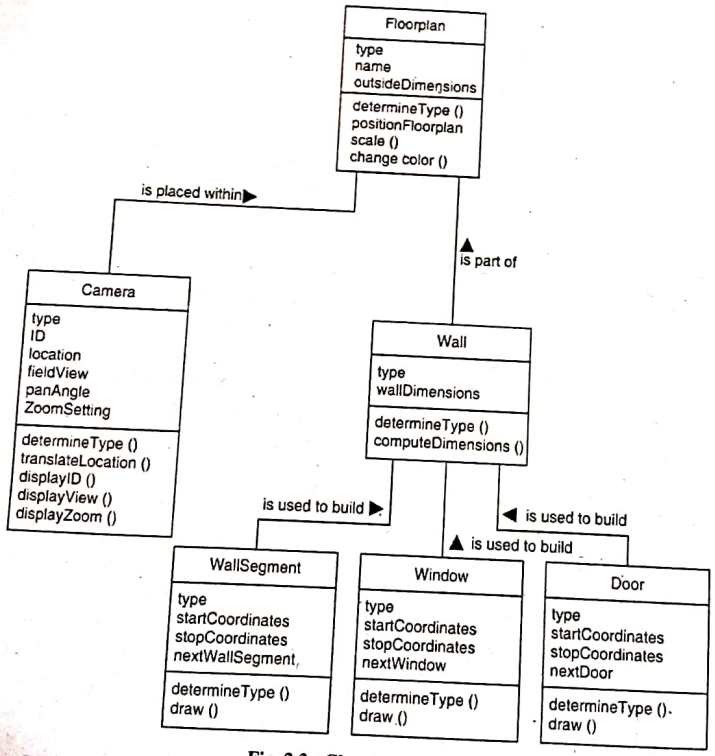


Fig. 2.3 : Class Diagram

Explain behavioral based model.

Ans. :

Behavioural Model

The behavioral model describes the way by which software will respond to any kind of external events or stimuli.

Following steps are necessary to create the model :

1. Evaluate all available use-cases to completely recognize the series of interactions within the system.
2. Identify events which force the interaction sequence and recognize how these events relate to specific objects.
3. Generate a sequence for all of the use-cases.
4. Develop a state diagram for the system.
5. Review the behavioural model to verify accuracy as well as consistency.

State Representations

In the context of behavioral modeling, there are two important characterizations of states which should be considered :

1. The states of all the respective classes as the system performs its function and
2. The state of the system identified from the outside as the system carry out its function.
3. The state of a class considers both passive as well as active characteristics.
4. A *passive state* is nothing but the current status of all the attributes of an object.
5. The *active state* of an object represents the current status of the respective object as it undergoes a continuing transformation or processing.

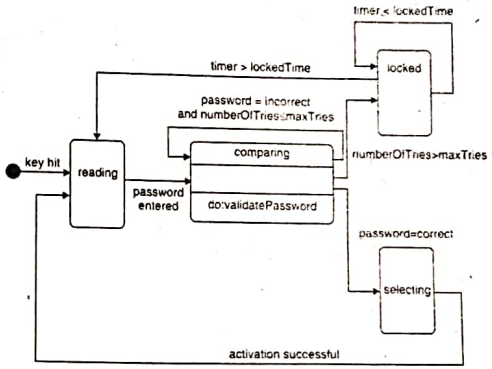


Fig. 2.4 : State Representation

The States of a System

1. **State :** It is a set of visible circumstances which characterizes the behavior of a system at a given time.
2. **State transition :** It is the movement from one state to another.
3. **Event :** It is an occurrence that causes the system to exhibit some predictable form of behaviour.
4. **Action :** It is a process that occurs as a consequence of making a transition.

Chapter 3 : Project Scheduling and Tracking

Q. 1 Explain 3 P's in software project spectrum.

Ans. :

Management Spectrum, 3Ps (People, Product and Process)

The management spectrum describes the management of a software project or how to make a project successful. Its focus is on the three P's; People, Product and Process. Here, the manager of the project has to control all these P's to have a smooth flow in the project progress and to reach the goal.

1. The People

People of a project includes from manager to developer, from customer to end user. But mainly people of a project highlight the developers.

It is so significant to have highly skilled as well as motivated developers that a PM-CMM (People Management Capability Maturity Model) is developed by the Software Engineering Institute to enhance the readiness of software enterprises to handle more and more robust applications by assisting to attract, grow, motivate, deploy, and retain the talent necessary for the improvement of their software development capability. Enterprises which attain high levels of maturity in the area of people management have a great probability of carrying efficient software engineering practices.

2. The Product

Product is nothing but any software which has to be designed and developed. For successful development of product there is need to set product objectives and scope, there must be consideration of substitute solutions, and also identification of technical and management constraints must be done. If there is lack of this information, it is not possible to define reasonable as well as correct estimates of the cost, an efficient estimation of risk, a practical breakdown of project tasks or a controllable project schedule which provides a meaningful indication of progress.

3. The Process

A software process is used to provide the framework from which it is possible to establish a complete plan for software development. There are different elements such as different tasks, milestones, work products, as well as quality assurance points which help to enable the framework regarding activities to be made compatible with the functionalities of the respective software project and the related necessities of the project team. At last, umbrella activities handle the process model. Umbrella activities are not dependent on any one framework activity and are carried out throughout the whole process.

Q. 2 Explain Software Project Estimation in detail.

Ans. :

Software Project Estimation

Measurements in the physical world can be divided into two categories : direct measures (e.g., the weight of a product) and indirect measures (e.g., the "quality" of product). In the same manner, it is possible to categorize software metrics. In the direct measures of the **software** process the elements involved are cost and efforts applied. Whereas in the direct measures of the **product**, the elements involved are LOC (lines of code) generated, speed of execution, memory size, and bugs recorded over some particular time span.

In the indirect measures of the product the elements involved are functionality, quality, complexity, efficiency, reliability, maintainability, and several other capabilities. The cost and effort necessary to develop software, the number of lines of code generated; and other direct measures are comparatively simple to gather, as long as particular conventions for measurement are set in advance. However there is difficulty in assessing and measuring directly the quality as well as functionality of software or its efficiency or maintainability. When the software metrics domain is partitioned into process, project, and product metrics, then it is observed that product metrics which are private to an individual are usually integrated to develop project metrics which are public to the respective software team.

The consolidation of Project metrics is done to generate process metrics which are public to the software firm as a whole. But what will be the way for an organization to combine metrics which are coming from different individuals or projects ? Individuals on 2 separate project teams record and make categorization of all the errors which are encountered during the software process. Those individual measures can be further combined to form team measures.

Team X encounters hundred errors during the software process before release. Team Y encounters seventy errors. Being everything is similar, which team is more efficient in searching errors throughout the process? Since we do not have knowledge of size or complexity of the projects, we cannot provide answer to this question. However, if we do normalization of the measures, it is probable to generate software metrics which enable comparison to broader organizational averages.

Q. 3 Explain size oriented software engineering metrics. **May 15, Dec. 16, May 17**

Ans. :

Metrics for Size Estimation

Size oriented software metrics are in general derived by the process of normalizing quality and/or productivity measures by assuming the software size which has been produced. If a software organization maintains simple records, a table of size-oriented measures will be as shown in Table 3.1. In Table 3.1, the details shown about all the software projects are developed in last five years.

Project alpha : 12,100 lines is the total size of code with 24 person-months of effort at a price of $168,000.

Table 3.1

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

It is important that the effort and cost recorded in the table does not represent only coding, but it also represents various software engineering activities such as analysis, design and testing. The further information shows that the size of documentation is of 365 pages. Before the release of software the count of errors was 134. Within the first year after release to the customer, the count of defects was 29. The number of people involved in development was three. For the purpose of metrics development which can be assimilated with other same types of metrics from various projects, as our normalization value, the Line of code will be selected.

As per the table information, a set of simple size-oriented metrics can be generated for all the projects :

1. Errors per KLOC (Thousands line of code)
2. Defects per KLOC,
3. $ Per KLOC,
4. Pages of documentation per KLOC

The Size-oriented metrics are not considered as the best solution for measuring the software process.

Q. 4 Explain Function Point Estimation Technique in detail.

Ans. : **May 17**

Function Points (FP)

As a normalization value, a measure of the functionality which is given by the application is taken by the Function-oriented software metrics. It is not possible to measure the functionality directly. Hence using other direct measures, it is necessary to derive it indirectly. The concept of Function-oriented metrics was proposed initially by Albrecht. A measure called the function point is suggested by him. Function points are basically derived with the help of an empirical relationship depending upon countable (direct) measures regarding the information domain of software and software complexity assessments.

Five information domain characteristics are determined and counts are provided. These are as follows :

1. Number of user inputs : Each user input should be considered.
2. Number of user outputs : Reports, screens, error messages, etc.
3. Number of user inquiries : An on-line i/p which results on on-line o/p.
4. Number of files : Each logical master file is counted.
5. Number of external interfaces : Data files on storage media.

Once these data have been collected, a complexity value is associated with each count. Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

Computing function points

Measurement parameter	Count		Weighting factor: Simple	Average	Complex		
Number of inputs		x	3	4	6	=	
Number of user outputs		x	4	5	7	=	
Number of user inquiries		x	3	4	6	=	
Number of files		x	7	10	15	=	
Number of external interfaces		x	5	7	10	=	
Count total							

Fig. 3.1

To compute function points (FP), the following relationship is used :

$$FP = \text{count total } [0.65 + 0.01\ \Sigma(Fi)]$$

(where count total is the sum of all FP entries)

The Fi (i = 1 to 14) are "complexity adjustment values" depending upon responses to the following given fourteen questions :

1. Does there is need of reliable backup and recovery to the system?
2. Is there any requirement of communications?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will it is possible to execute system in current, greatly utilized operational environment?
6. Does there is need of on-line data entry to system?
7. Does there is need of the input transaction to on-line data entry so as to build over multiple screens or operations?
8. Is the updation of master files is done on-line?
9. Is there is complexity in the inputs, outputs, files, or inquiries?
10. Is there is complexity in internal processing?
11. Can the code designed be reusable?
12. Is there is involvement of conversion and installation in the design?
13. Is the system designed for more than one installation in various organizations?
14. Is the design of application facilitates change and ease of use by the user?

Now all these questions are usually answered with the help of a scale in the ranges 0 – 5. The constant values which are present in the equation and the weighting factors which are applied to the information domain counts are decided based on the facts.

As soon as the calculation of function points is completed, they are used so as to normalize measures for the different factors such as software productivity, quality, and other attributes like Errors per FP, Defects per FP, $ per FP, Pages of documentation per FP, FP per person-month.

Q. 5 Explain COCOMO Model. **May 15, Dec. 16**

Ans. :

COCOMO II Model

The COCOMO (Constructive Cost) model is an empirical model that was derived with the help of gathering data from a large number of software projects. These data were analyzed to determine formulae that are the best fit to the observations. These formulae link the size of the system and product, project and team factors to the effort to develop the system.

We can select to use the COCOMO model for number of reasons :

1. It is well documented, accessible in the public domain and supported by public domain and commercial tools.
2. It has been widely used and evaluated in several organizations.

COCOMO 81, first version of the COCOMO model was a three-level model. The first level presents an original rough estimate. The second level modifies this using a several project and

process multipliers; and the most detailed level creates estimates for different stages of the project.

Table 3.2 shows the basic COCOMO formula for different types of projects. The multiplier M reflects team, process and project characteristics.

Table 3.2

Project Complexity	Formula	Description
Simple	PM = 2.4 (KDSI)1.05 × M	Well-understood applications produced by small teams.
Moderate	PM = 3.0 (KDSI)1.12 × M	More complex projects where team members may have partial experience of related systems.
Embedded	PM = 3.6 (KDSI)1.20 × M	Complex projects where the software is element of a strongly coupled.

COCOMO 81 thinks that the software would be developed according to a waterfall process using standard programming languages like C. To take these modifications into account, the COCOMO II model identifies different approaches to software development like prototyping. COCOMO II helps in development of a spiral model and embeds number of sub-models that create the whole estimates. Fig. 3.2 shows COCOMO II sub-models as well as where they are used.

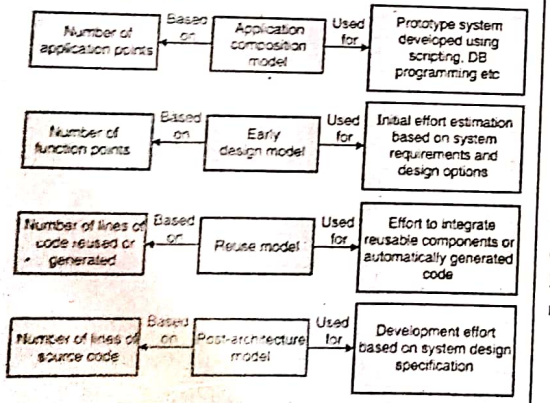


Fig. 3.2 : The COCOMO II model

1. The application-composition model

It supposes that systems are generated from reusable components, scripting or database programming. It is designed to create estimates of prototype development. The application-composition model was established into COCOMO II to maintain the estimation of effort needed for prototyping projects. It depends on an estimate of weighted application points also called as object points separated by a standard estimate of application-point productivity.

The estimate is then changed according to the complexity of development of every object point. Programmer productivity is also based on the experience and capability as well as the capability of the CASE tools used to support development.

Table 3.3 shows the levels of object-point productivity.

Table 3.3 : Object point productivity

Developer's experience and capability	Very low	Low	Nominal	high	Very high
CASE maturity and capability	Very low	Low	Nominal	high	Very high
PROD (NOP/month)	4	7	13	25	50

Therefore, the concluding formula for effort calculation for system prototypes is :

$$PM = (NAP \times (1 \times \%reuse/100)) / PROD$$

Where,

PM - effort estimate in person-months.

NAP - total number of application points in the delivered system.

%reuse - estimate of the amount of reused code in the development.

PROD - object-point productivity

2. The early design model

This model is used at the time of early stages of the system design after the requirements have been established. Estimates are depending on function points, which are then changed to number of lines of source code. Estimates can be completed after the user requirements have been agreed.

Depend on standard formula for algorithmic models.

$$Effort = A \times SizeB \times M$$

The multiplier M in COCOMO II is depends on a simplified collection of seven projects and process characteristics that influence the estimate. These can increase or decrease the effort needed. These characteristics used in the early design model are product reliability and complexity (RCPX), reuse required (RUSE), platform difficulty (PDIF), personnel capability (PERS), personnel

experience (PREX), schedule (SCED) and support facilities (FCIL).

This results in an effort computation as follows :

$$PM = 2.94 \times SizeB \times M$$

Where,

$$M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$$

3. The reuse model

Reuse model is used to calculate the effort needed to integrate reusable software or program code that are automatically created by design or program translation tools.

For COCOMO II, reused code is of 2 types. First is the black-box code which can be reused without having knowledge of code or making changes in it. The effort regarding development for **black-box code** is taken to be zero.

Code that has to be adapted to combine with new code is called **white-box code**. Some development effort is needed to reuse this because it has to be known and changed before it can work accurately in the system.

The formula for effort estimation is :

$$PMAuto = (ASLOC \times AT/100) / ATPROD$$

Where,

AT - Percentage of adapted code that is automatically created

ATPROD - Productivity of engineers in integrating such code.

For example

If there is a total of 10,000 lines of white-box reused code in a system and 40% of this is automatically generated, and ATPROD is 2400 then the effort required to integrate this generated code is :

$$(10{,}000 \times 40/100) / 2400 = 1.67 \text{ person months}$$

The other component of the reuse model is used when a system contains some new code and some reused white-box components that have to be integrated. Therefore, if 30,000 lines of code is to be reused, the new equal size estimate might be 6,000. Essentially, reusing 30,000 lines of code is engaged to be equivalent to writing 6,000 lines of new code.

This computed form is added to the number of lines of new code to be developed in the COCOMO II model.

1. **ASLOC :** Number of lines of code in the components that have to be adapted;
2. **ESLOC :** Equivalent number of lines of new source code.

The below formula is used to compute the number of equal lines of source code :

$$ESLOC = ASLOC \times (1 \times AT/100) \times AAM$$

Where,

AAM is nothing but Adaptation Adjustment Multiplier.

Q. 6 Explain different specialized Estimation Techniques.

Ans. :

Specialized Estimation Techniques

In general the decomposition techniques can be used for any software project.

However, when there is very short period of time for project completion (weeks instead of months) which is probable to have a continuing series of changes, estimation in particular should be abbreviated.

Estimation for Agile Development

Since the requirements regarding an agile project are set by user scenarios, an estimation approach can be developed which is informal, reasonably disciplined, and meaningful in the environment of project planning for each software increment.

Approach of Decomposition is used in the estimation for agile projects which encompasses the following steps :

1. Each and every user scenario is considered independently for estimation purposes.
2. The scenario is divided into a group of software engineering tasks which will be necessary to develop it.

3a. The estimation of effort needed for each task is done separately.

3b. On the other hand, the estimation of "volume" of the scenario can be done in LOC (Line of Code), FP (Function Point), or in any other volume-oriented measure (e.g., use-case count).

4a. Estimates of all the tasks are combined to set an estimate for the scenario.

4b. Alternatively, historical data is used to translate the volume estimate for the scenario into effort.

5. The effort estimates for each and every scenario which is to be implemented for a provided software increment are added to set the effort estimate for the increment.

Since the project duration essential for the software increment development is relatively short (normally 3 to 6 weeks), two purposes are served by this estimation :

(1) To make sure that the number of scenarios which are to be considered in the increment conforms to the existing resources, and

(2) To set a basis for assigning effort as the increment is developed.

Estimation for WebApp Projects

WebApp projects usually like to refer the agile process model. An updated FP measure helps to develop an estimate for the WebApp.

Following approach is suggested when adapting FP for WebApp estimation :

1. Inputs are nothing but the input screen or form (such as CGI or Java), every maintenance screen, each tab (if used).
2. Outputs are every static Web page, every dynamic Web page script (such as ASP, PHP, or other DHTML script), and every report.
3. Tables are every logical table in the DB plus, every XML object (if used).
4. Interfaces preserve their definition as logical files (such as unique record formats) into the out-of-the-system boundaries.
5. Queries are the one which are published externally or take help of a message-oriented interface such as DCOM or COM external references.
6. Function points (FPs) are considered as logical indicator of volume for a WebApp.

It is possible to determine volume of a WebApp by gathering measures (known as "predictor variables") allied with the application (such as page count, media count, function count), its characteristics of Web page (such as page complexity, linking complexity, graphic complexity), characteristics of media (such as media duration), and functional characteristics (such as code length, reused code length).

These measures help to establish empirical estimation models for whole project effort, page and media authoring effort, and scripting effort.

Q. 7 Write meaning of CPM? Explain with example.

Ans. :

Critical Path Method

In a project, various activities are executed in parallel by different teams. Some activities are dependent on others and hence cannot start before another task is completed. For example, team A cannot test database connectivity unless team B updates the same database. When all the activities along with dependencies are documented, one can determine the activity that would consume the longest duration. Considering this longest duration, all other activities can be executed in parallel with this activity (An activity which takes longest duration to complete). Critical Path is longest sequence of activities in a project plan which must be completed on time for the project to complete on due date.

Based on which logic has been used in the project, some project can have one critical path and some can have many critical paths. If the delay occurs in any of the activity involving the critical path then it will ultimately leads to the delay in the project deliverables. If such condition occurs then re-sequencing is done so that the project deadlines can be achieved. Critical path is considered as a method which is based on mathematical calculations that are used for the purpose of scheduling project activities.

Critical path method was first introduced in 1950s as a joint venture between Remington Rand Corporation and DuPont Corporation. Initially critical path method was used for managing plant maintenance projects. The original critical path method was developed for construction project. Now the CPM can be used in any project where interdependent activities are involved. In the critical path method, the critical activities of a project are first identified.

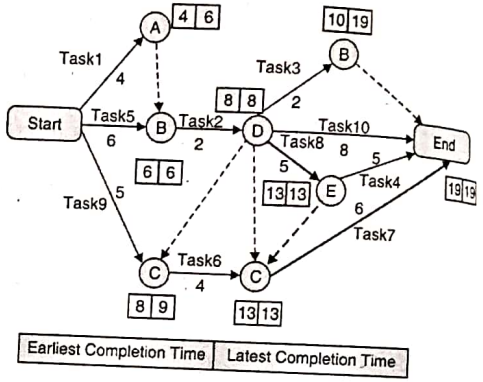


Fig. 3.3 : Critical Path

Following are the Steps in Critical Path Method.

There are generally 5 steps involved the CPM :

1. Activity specification

Work Break Down structure (WBS) is used to find out which activities are involved in the project and hence it is the main input of the CPM.

While specifying the activity, generally the higher-level activities are selected for critical path method. If the detailed activities are used, the CPM may become more complex to understand and maintain.

2. Activity sequence reorganization

The correct activity sequence is recognized.

Following are the steps to recognize the sequence of the CPM :

1. Identify the task that takes place before the critical task happens.
2. Identify the tasks that should be completed at the same time.
3. Identify the tasks that should happen immediately after critical task.

3. Network diagram

Once the activity sequence is identified, the network diagram can be drawn.

4. Estimates for each activity

This is the direct input from the WBS based estimation sheet.

5. Identification of the critical path

For this, there is need of determining 4 parameters of each activity of the network.

1. Earliest start time (ES) – When the earlier dependent activities are completed, one can start the earliest time of an activity.
2. Earliest finish time (EF) - ES + activity time.
3. Latest finish time (LF) - The most recent time of an activity can be completed without having delay in the project.
4. Latest start time (LS) - LF - activity duration.
5. Float Time - ES-LS

During the float time, an activity can be delayed without having delay in the project date. The critical path is considered as the longest path of the respective network diagram. There is impact of the activities involved in the critical path on the deadline of the project. If there is delay in an activity of this path, then the project will also be delayed. If the project management needs to gear up the project, activities involved in the critical path should be reduced.

Advantages of Critical Path Method

1. Visual representation of the project activities.
2. To calculate Project deadlines.
3. To track the critical activities.

Q. 8 Explain PERT.

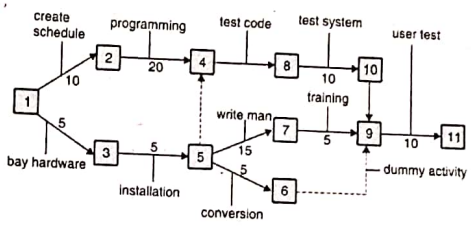
Ans. :

Program Evaluation and Review Technique (PERT)

PERT is related to management probabilities. PERT methods is included in many simple statistical methods. A PERT is a project management technique used to plan, organize, and coordinate tasks during the project. PERT is for Program Evaluation Review Technique, which was developed by the U.S. Navy in the 1950s for the purpose of managing the Polaris submarine missile program.

A PERT chart is a graphical representation of project network diagram that will consist of numbered nodes. These nodes represent events. The project is linked by the directional line which represents the tasks in the project. The direction of the arrow on the diagram represents the sequence of the task. In the Fig. 3.4, the tasks between nodes 1, 2, 4, 8, and 10 have to be completed in series. These are called as dependent or serial tasks. The tasks between nodes 1 and 2, and nodes 1 and 3 are not dependent on each other and hence they can be started simultaneously. These tasks are called parallel or concurrent tasks.

Tasks which can be carried out in sequence but doesn't need any resources or completion time are called as an event dependency. These are denoted with the help of dotted lines with arrows and are known as dummy activities. For example, the dashed arrow between nodes 5 and 4 is termed as dummy activity.



Numbered rectangles are nodes and represent events or milestones. Directional arrows represent dependent tasks that must be completed sequentially. Diverging arrow directions (e.g. 1-2 and 1-3) indicate possibly concurrent tasks. Dotted lines indicate dependent tasks that do not require resources.

Fig. 3.4 : PERT

The Possibilities of the PERT are listed below :

There are three estimates involved in PERT

1. **TOPT**

   This is the fastest time during which activity can be finished. The assumption is made that all the required resources are available and all the previous activities are completed as planned.

2. **TLIKELY**

   Generally project managers are asked to present one estimate during the initial project period. In that case, this is the estimate which goes to the upper management.

3. **TPESS**

   This is the maximum time required to complete an activity. It is assumed that many things related to project activity will go wrong. A lot of rework will need to do and resource unavailability is considered when such estimation is derived.

Q. 1 Explain qualities of good software design and design principles.

Ans. :

Qualities of Good Design

There are number of qualities of good design :

(1) Innovative

Innovative design can be either completely new design or redesign of existing product. New design gives unseen value to market where as redesign improves the quality of an existing product.

(2) Functional

Good design fulfils all its intended functions to solve user's problem. It focuses on usefulness of a product by optimizing its functionality.

(3) Honest

A good design is honest. An honest design expresses the functions and values it offers. It never attempts to modify Organiser's and User's view with promises it can't keep.

(4) User - oriented

Good design is developed based on its use and intended to improve solution to problem for its user.

User-oriented design gives intellectual as well as material value to system which in turn achieves user's satisfaction.

(5) Correctness

Correctness is an important quality of good design. A good design should correctly achieve all required functionalities as per SRS document.

Design Principles

Following are important design principles :

1. The design process should not suffer from 'tunnel vision'.
2. The design should be traceable to the analysis model.
3. The design should not reinvent the wheel.
4. The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.
5. The design should exhibit uniformity and integration.
6. The design should be structured to accommodate change.
7. The design should be structured to degrade gently, even when unusual data, events, or operating conditions are encountered.
8. Design is not coding, coding is not design.
9. The design should be assessed for quality as it is being created, not after the creation.
10. The design should be reviewed to minimize conceptual (semantic) errors.

Q. 2 What is cohesion? Explain different forms of it.
May 15, Dec. 15, May 16, Dec. 16, May 17, Dec. 17, May 18

Ans. :

Cohesion

Cohesion define the degree to which components of system are related to each other i.e. it measures the strength of relationship between two components of system.

Good system design must have high cohesion between the components of system.

Different Types of Cohesion

(i) Co-incidental cohesion

An unplanned cohesion which results into decomposition of a program into smaller components for the modularization of system is called co-incidental cohesion.

Typically these types of cohesions are never accepted.

(ii) Logical cohesion

When logically classified elements are combined into a single module then it is called as logical cohesion. Here all elements of modules contribute to same system operation.

(iii) Temporal cohesion

Temporal cohesion occurs when component of a system performs more than one function and these functions must occur within same time span. It is generally found in initiation and termination activities.

(iv) Procedural cohesion

When components of system are related to each other only by sequence then there exists procedural cohesion. Loop in programming language is good example of procedural cohesion.

(v) Communication cohesion

When elements of module perform different functions but each function accepts same input and generates same output then that module is said to be communicational cohesive.

It is sometimes acceptable if no other alternative is easily found.

(vi) Sequential cohesion

A module is said to be sequentially cohesive if its functions are related in a way such that output of one function acts as an input data to other function.

This type of cohesion is easily maintainable.

(vii) Functional cohesion

If elements of module are grouped together since they contribute to a single function then such a module is functionally cohesive module.

All elements of such a module are necessary for successful execution of function.

Advantages of Cohesion

(i) High cohesion among system components results in Better program design.

(ii) High cohesion components can be easily reused.

(iii) High cohesion components are more reliable.

Disadvantages of Cohesion

(i) Low cohesion components are difficult to maintain.

(ii) Low cohesion components cannot be reused.

(iii) Low cohesion components are difficult to understand.

(iv) Low cohesion components are less reliable.

Q. 3 What is coupling? Explain different forms of it.

May 15, Dec. 15, May 16, Dec. 16, May 17, Dec. 17, May 18

Ans. :

Coupling

Coupling is an indication of strength of interconnection between various components of a system. Highly coupled components are more dependent on each other whereas low - coupled components are less dependent or almost independent on each other. Good design always have low coupling between components of system.

Example,

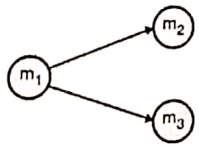


Fig. 4.1 : Coupling

Here module m_1 is dependent on modules m_2 and m_3 for its functionality i.e. if any one of m_2 and m_3 fails m_1 will also fail. But m_2 and m_3 are free modules they do not depend on any other component for their functions. Thus success or failure of any other module does not affect modules m_2 and m_3. If modules are highly coupled with each other then change in any one of them force change in other modules or components. One of the solutions to reduce coupling between system components is a functional design.

In general, in system development coupling is always constructed with the cohesion.

Types of Coupling

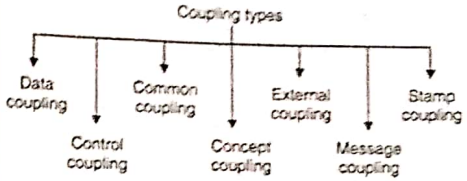


Fig. 4.2 : Coupling types

(i) Data coupling

Data coupling occurs between two components of system when they pass data to each other by parameters using argument list and each element in argument list is used.

A component becomes difficult to maintain if too many parameters are passed.

(ii) Control coupling

When data are passed between two components and that affects internal logic of a component then there exists control coupling between those two components. It passes the control flags between system components.

(iii) Common coupling

Common coupling is also called as global coupling since in these coupling components of system communicates with each other by using global area. Common coupling becomes difficult to track when data in global area is changed.

(iv) Content coupling

When one component refers to the internals of the other component then those two components are said to be content coupled. As content coupling is worst type of coupling it should not be used in designing.

(v) External coupling

If two components of system share an externally imposed data format, communication protocol or device interface then they are said to be externally coupled. External coupling refers to communication between system components and external tools or devices.

(vi) Message coupling

Message coupling occurs between two components of system when those components communicate with each other via message passing. In this coupling components are independent of each other; thus it is low type coupling.

(vii) Stamp coupling

Stamp coupling occurs between two components of system when data between them are passed by arguments using a data structure containing elements which may or may not be used.

Stamp coupling is also low type coupling.

Advantages of Coupling

(i) Low coupling components do not force ripple effect to other components.

(ii) Low coupled components can be reused.

(iii) With low coupling among components, system can built faster.

Disadvantages of Coupling

(i) High coupling components are difficult to understand.

(ii) High coupling components forces change in other components if one component changes.

(iii) High coupling slows down the development process.

Q. 4 What are benefits of high cohesion and low coupling ? **Dec. 17**

Ans. :

Benefits of high cohesion

1. **Readability :** (Closely) Related functions are contained in a single module.
2. **Maintainability :** Debugging tends to be contained in a single module.
3. **Reusability :** Classes that have concentrated functionalities are not polluted with useless functions.

Benefits of low coupling

1. **Maintainability :** Changes are confined in a single module.
2. **Testability :** Modules involved in unit testing can be limited to a minimum.
3. **Readability :** Classes that need to be analyzed are kept at a minimum.

Q. 6 Differentiate between cohesion and coupling.

Ans. :

Difference between Coupling and Cohesion

Parameter	Cohesion	Coupling
Concept	Cohesion indicates relationship with in the module.	Coupling indicates the relationship between two or more different modules.
Represents	Cohesion represents relative functional strength of a module.	Coupling represents relative interdependence among the modules.
Degree	It is a degree up to which module focuses on systems' single function/component.	It is a degree to which one module depends on another module.
High or Low	High cohesion is good for system design.	Low coupling is good for system design.

Q. 7 Explain different Architectural styles with suitable brief example of each. **Dec. 15**

Ans. :

Architectural Style

Architectural styles describe a set of systems which are internally linked with each other and share structural and semantic properties. In short, the purpose of using architectural styles is to set up a structure for all the components present in a system. If an existing architecture is to be re-engineered, then burden of an architectural style results in basic changes in the structure of the system. By applying certain restriction on the design space, we can make different style-specific analysis from an architectural style. A computer-based system (software is part of this system) demonstrates one of the several available architectural styles.

Every architectural style defines a system type that includes the following :

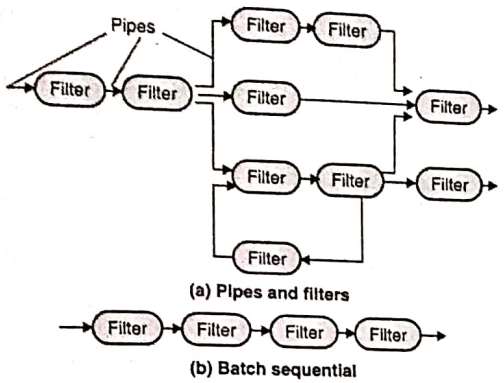
1. Computational components like clients, server, filter, and database to execute the required system function.
2. A collection of connectors like procedure call, events broadcast, database protocols, and pipes to offers communication between the computational components.
3. Constraints to describe integration of components to form a system.
4. A semantic model, which allow the software designer so as to recognize the characteristics regarding the system as a whole by the way of studying the characteristics of its components.

Following are the common architectural styles :

1. Data-flow Architecture

The purpose of data-flow architecture is to accept some inputs and transforms it into the required outputs by applying a sequence of transformations. Each component called as filter transforms the data and sends this transformed data to other filters for additional processing with the help of the connector called as pipe. Every filter operates as an independent entity, that is, it is not focused on the filter which is generating or consuming the data. A pipe is a unidirectional channel which transfers the data received on one end to the other end.

It does not change the data in anyway; it just delivers the data to the filter on the receiver end.



(a) Pipes and filters

(b) Batch sequential

Fig. 4.3 : Data flow architecture

Most of the times, the data-flow architecture reproduce a batch sequential system. In this system, a batch of data is accepted as input and then a sequence of sequential filters are applied to transform this data. Example of this architecture is UNIX shell programs. In these programs, UNIX processes act as filters and the file system by which UNIX processes communicates act as pipes.

Advantages of data-flow architecture

1. It supports maintainability, reusability and modifiability.
2. Concurrent execution is supported by this style.

Disadvantages of data-flow architecture

1. It frequently degenerates to batch sequential system.
2. The application which needs more user interactions is not supported by data-flow architecture.
3. It is not easy to coordinate two different but related streams.

2. Object-oriented Architecture

In this architectural style, the components of a system wrap data and operations into single unit, which are used to change the data. In this style, components are represented as *objects* and they communicate with each other through methods (connectors).

Characteristics of object-oriented architecture

1. Objects preserve the integrity of the system.
2. An object is not aware of other objects' representation.

Advantages of object-oriented architecture

1. It allows designers to divide a problem into a group of independent objects.
2. The implementation detail of objects is known to other object so that they can be changed without affecting other objects.

3. Layered Architecture

In layered architecture, every layer is responsible for performing a well-defined set of operations. These layers are organized in a hierarchical manner, where every layer is built upon one below another.

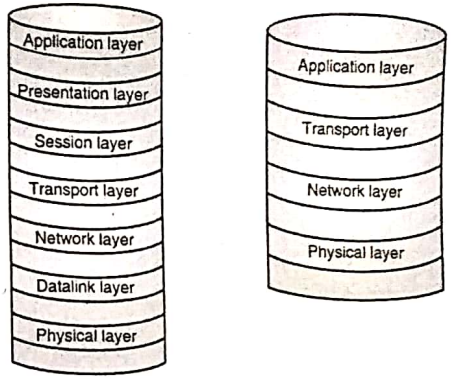


Fig. 4.4 : OSI and Internet Protocol Suite

Each layer offers a set of services to the layer which is located above it and acts as a client to the layer which is located below it. The interaction among layers is provided through protocols that describe a set of rules to be followed throughout the interaction. One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organization for Standardization) communication system.

4. Data-centered Architecture

A data-centered architecture has two unique components :

1. Central data structure
2. Collection of client software

A central data structure is also known as data store.

The data store such as database or a file demonstrates the current state of the data where as the client software performs various operations such as add, delete, update, etc. on the data which is stored in the data store. Sometimes the data store permits the client software to access the data independent of any changes or the actions of other client software.

In this architectural style, new components related to clients can be added and existing components can be changed easily which does not require change in other clients. This is because client components work without depending upon any other components. A variation of this architectural style is blackboard system in which the data store is transformed into a blackboard that informs the client software when the data is modified.

Additionally, the information can be transferred between the clients with the help of blackboard components.

Advantages of the data-centered architecture

1. Data repository is not dependant of the clients.
2. Clients work independent of each other.
3. Adding new clients can be easy.
4. Modification can be very easy.
5. It accomplishes data integration in component-based development with the help of blackboard.

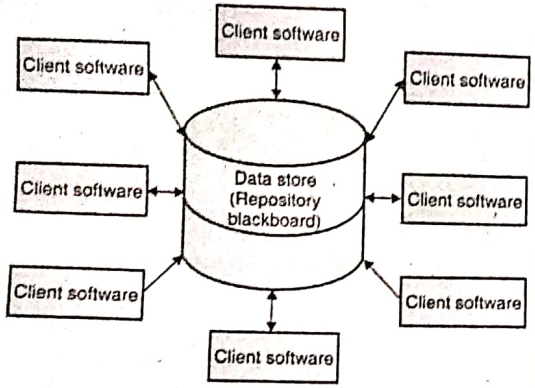


Fig. 4.5 : Data-centered architecture

5. Call and Return Architecture

A call and return architecture allows software designers to get such a program structure where modification is easy.

This architecture style contains following two sub-styles.

A. Main program/subprogram architecture

In this style, function is divided into a controlled hierarchy where the main program contains and executes a number of program components, where as the contained components may in turn executes other components.

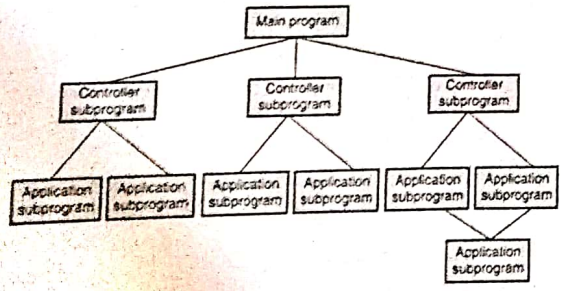


Fig. 4.6 : Main program/subprogram architecture

B. Remote procedure call architecture

In this style, components of the main program or subprogram architecture are distributed over a network across number of computers which can be accessed remotely.

Q. 8 Write short note on User Interface Design.

Dec. 16, May 18

Ans. :

User Interface Design

User interface is considered as a front-end application view which is interacted by the user so as to use the software. User is able to manipulate as well as control the software and hardware with the help of user interface. Now-a-day, user interface appears in each and every place where there is existence of digital technology, right from desktops, smart phones, cars, music players, airplanes, ships etc. User interface is an integral component of software and is designed in such a manner that it should be able to provide the user insight of the software.

UI offers a fundamental platform for the communication of user and computer. The form of UI can be graphical, text-based, audio-video based. It is usually depends upon the underlying platform (hardware and software combination).

The popularity of software depends upon following aspects :

1. Attractive
2. Simple to use
3. Responsive in short time
4. Clear to understand
5. Consistent on all interfacing screens

UI is broadly divided into two categories :

1. Command Line Interface
2. Graphical User Interface

Command Line Interface (CLI)

CLI was considered as a great tool of communication with computers in previous days. Now-a-days also CLI is the first choice of number of technical users as well as programmers. CLI is considered as minimum interface which software can provide to its users. A command prompt is provided by the CLI which is a place where the user writes instructions and feeds to the system.

Here there is need for the user to remember the syntax of command and its use. In previous days CLI were not programmed to handle the user errors effectively. A command is considered as a text-based reference to set of instructions, which should be executed by the respective system. There are several methods such as macros, scripts which make it simple for the user to operate. Very less amount of computer resource is used by the CLI in comparison with GUI.

There are following elements in a text-based command line interface :

1. **Command Prompt :** It is text-based notifier which usually displays the context where the user is working. Software system generates it.
2. **Cursor :** It is a small horizontal line or a vertical bar with the height of line. It is used to represent position of character at the time of typing. Cursor is usefully found in blinking state. It moves when user writes or deletes text.
3. **Command :** A command is nothing but an executable instruction. There may be one or more parameters to it. Output of command is displayed inline on the screen. After displaying output, command prompt is set to the next line.

Graphical User Interface (GUI)

Graphical User Interface offers graphical means to the use for interacting with the system. GUI can be combination of both hardware and software. Using GUI, user is able to interpret the software. In general GUI is considered as more resource consuming as compared to CLI. With the help of latest advanced technology, the programmers and designers are able to create complex GUI designs which perform with better efficiency, accuracy and speed.

Q. 9 What are the features of good user interface? **May 15**

Ans. :

User Interface Golden Rules

Designers need to solve problems every day, and finding the right solution involves in-depth research and carefully planned testing. The following rules are known as the golden rules for GUI design.

1. Strive for Consistency

Users should not have to wonder whether different words, situations, or actions mean the same thing. Do not confuse your user keep *words and actions* consistent.

Use "The Principle of Least Surprise".

Example : Car Climate Control

In other words, use all elements across your application consistently. For example a certain style of button should always do the same thing, or navigation should function logically, going deeper in hierarchy.

Consistency of :

1. Workflow / Processes
2. Functionality
3. Appearance
4. Terminology

2. Visibility of system status or Offer informative feedback

The system should always keep users informed about what is going on. Through appropriate feedback in a *reasonable* time. Don't keep the users guessing tell the user what's happening.

Example : OS Installation status.

The user wants to be in control, and trust that the system behaves as expected, It could be even said that users don't like surprises.

For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

Feedback

1. Relevant
2. Fits importance and urgency
3. Comprehensible and meaningful
4. Within appropriate context (time and place)

3. Match between system and the real world or Design dialog to yield closure

Again, the less the users have to guess the better. The system should speak the users' language (use words, phrases and concepts familiar to the user), rather than special system terms.

Example : Payment proceeding (by Ramakrishna)

Sequence of actions should be organized into groups with a beginning, middle and end. When a process is finished, remember to display a notification message.

Let the user know that he has done all that's needed.

Design

1. Grouping of actions.
2. Explicit completion of an action.
3. Well-defined options for the next step.

4. User control and freedom or Permit easy reversal of actions

Shneiderman puts it nicely "This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options."

Example : Document History

In applications, this refers to the undo and redo functionality. Clearly mark an "*emergency exit*" to leave the unwanted state without having to go through an extended dialogue.

Reversal of actions

1. No interference with workflow
2. More freedom for the user
3. Single-action undo and action history

5. Error Prevention and Simple Error Handling

Users hate errors, and even more so hate the feeling that they themselves have done something wrong. Either *eliminate* error-prone conditions or *check for them* and notify users about that before they commit to the action.

Example : Password Enter

As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer a simple, comprehensive mechanism for handling the error.

Error Prevention

1. Error prevention over error correction
2. Automatic detection of errors
3. Clear error notifications
4. Hints for solving the problem

6. Reduce short-term memory load or Recognition rather than recall

As Nielsen says, recognizing something is easier than remembering it. Minimize the user's memory load by making objects, actions, and options *available*. The user should not have to remember information from one part of the dialogue to another. Instructions should be *visible*.

Example : Ring/Silent switch

Use iconography and other visual aids such as themed coloring and consistent placement of items to help the returning users find the functionalities.

Reduce memory load

App has a clear structure

1. "Recognition over recall"
2. Implicit help
3. Visual aids

7. Enable frequent users to use shortcuts

Allow users to *tailor* (manipulate and personalize) frequent actions. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

Example : Click Commands and Shortcuts

Shortcuts

1. Keyboard shortcuts
2. "Power User" features
3. Action automation

8. Aesthetic and Minimalist design

Minimalist doesn't mean limited. All information should be *valuable* and *relevant*.

Example : Simple Interface.

Simplify interfaces by removing unnecessary elements or content that does not support user tasks.

9. Help users recognize, diagnose, and recover from errors

Error messages should be expressed in *plain language* (don't use system language to describe what the system is doing), precisely *indicate* the problem, and constructively *suggest* a solution.

Example : Plain Message

Tell the user clearly and plainly what's happening both on the background and when they perform an action.

10. Help and documentation

Even though it is better if the interface can be used without documentation, it may be necessary to provide help and documentation. Any help information should be *easy to search, focused* on the user's task, list of *concrete steps*, and *not too large*.

Chapter 5 : Software Risk, Configuration Management & Quality Assurance

Q. 1 Write short note on Risk Management.

May 15, Dec. 15, Dec. 16, May 18

Ans. :

Risk Management

Risk *is the possibility of suffering loss. In the project development, the loss illustrates the impact to the project which could be in the form of diminished quality of the end product, increased costs, delayed completion, or failure.*

Fig. 5.1 : Kinds of risks

Management are as follows :

1. Schedule / Time-Related / Delivery Related Planning Risks

These risks are related to running behind schedule and are essential time-related risks, which directly impact the delivery of the project.

Some of the reasons for such risks are :

1. Incorrect Time Estimation, and consequently an incorrect project schedule.
2. Improper Resource Allocation.
3. Underutilization of Resources.
4. Superficial Understanding of Project Complexities.
5. Unexpected Expansion of Project Scope.

Incorrect time estimation may occur because activities may have external dependencies such as client approvals, subcontractors etc. and a delay in a critical path activity has a cascading effect on the entire project. Resource Allocation may be improper / underutilization of resources may take place, especially if resources are shared between projects. A silo approach of members in various teams in a project may lead to an isolated superficial understanding of project complexities which may result in delays in subsequent stages e.g. when different development teams work on various aspects of a software project and run into issues during system/integration testing.

2. Budget / Financial Risks

These are the monetary risks which are associated with budget overruns.

Some of the reasons for such risks are :

1. Improper Budget Estimation.
2. Cost Overruns due to underutilization of resources.
3. Expansion of Project Scope.
4. Improper Tracking of Finances.

Underutilization of resources especially happens when resources are shared between projects because it becomes difficult to effectively manage such resources and a certain amount of productivity may go waste. Further, unexpected expansion of project scope (due to addition of features by clients, etc) may lead to budget overruns as such expansions may not have been factored into the original estimates. Delay of projects may also have certain penalty costs associated with them e.g. construction projects.

3. Operational / Procedural Risks

These are risks which are associated with the day-to-day operational activities of the project.

These could be due to any of the below reasons :

1. Improper Process Implementation.
2. Silo approach followed by software development teams leading to conflicts.
3. Conflicting Priorities.
4. Lack of conflict resolution / team spirit.
5. Lack of clarity in responsibilities.
6. Breakdown in communications.
7. Lack of sufficient training.

Effective team communication is an essential part of project management and in people-intensive projects such as software projects, there is a strong need for an established communication structure, a setup for escalation, a conflict resolution process, established project priorities and above all, the employees need to be trained in making use of these processes within the organization.

4. Technical / Functional / Performance Risks

These are technical risks associated with the functionality of the software or with respect to the software performance. In order to compensate for excessive budget overruns and schedule overruns, companies sometimes reduce the functionality of the software. Software testing is a downstream stage in the software development lifecycle and as the project falls behind schedule, downstream activity times are shrunk in order to meet delivery dates which results in insufficient software testing.

Further, developers face a constant trade-off between achieving maximum functionality of the software (in terms of software features) and peak performance (maximum speed and quick response time by minimizing and eliminating unnecessary add-ons from the software). In order to maintain the purity of the software development process, while simultaneously catering to the customers' needs, a mutually agreed-upon cut-off date should be determined, beyond which "expected software functionality" would be frozen and any further requirements would be handled in subsequent software's releases.

5. Other Unavoidable Risks

All the risks described above are those which can be anticipated to certain extend and planned for in advance. However there are certain risks which are unavoidable in nature.

The reasons for such unavoidable risks are described below.

1. Changes in government policy,
2. Obsolescence (becoming outdated) of software due to new technology from a rival company,
3. Loss of contracts due to changes at customers end.

Risk Management is a software engineering practice with processes, methods, and tools for managing risks in a project. It provides a disciplined environment for proactive decision-making to :

Know continuously what can go wrong (risk identification and assessment), Determine what risks are important to deal with (risk prioritization), Implement strategies to deal with those risks (risk resolution). Risk Analysis is series of procedures such as risk management planning, analysis of risks, identifying and controlling the risk involved in project.

Risk Management Procedure basically includes following activities :

1. Risk Assessment
2. Risk Identification
3. Risk Containment

Q. 2 Write note on RMMM. **May 15**

Ans. :

RMMM

The main aim of all of the risk analysis related activities is to guide the project team in forming a strategy for dealing with risk. In an effective strategy there are three important issues : **risk avoidance (mitigation)**, **risk monitoring**, and **risk management and contingency planning**. If proactive approach to risk is accepted by a software team, avoidance is considered as always the best strategy. This is accomplished by setting a plan for risk mitigation.

For example, consider that high staff turnover is marked as a project risk r_1. As per previous history and management instinct, the likelihood l_1 of high turnover is guessed to be 0.70 (70 percent, rather high) and the impact x_1 is projected as critical. It indicates that there will be critical impact of high turnover on project cost and schedule. To mitigate this risk, a strategy can be developed to decrease turnover.

Among the possible steps to be taken are :

Communicate with current staff to decide reasons for turnover (e.g., poor working conditions, low pay, competitive job market). Mitigate these reasons that are controllable prior to the project starts. After commencement of project, assume there will be occurrence of turnover and develop techniques to take care of continuity when people leave. Organize project teams to widely disperse the information regarding each development activity.

Define work product standards and set system to assure that all the respective models and documents are developed in expected time duration. Allocate a backup staff for all the critical technologists. The factors are **monitored** by the project manager which may give a signal of whether the risk is becoming more or less likely. In the case when there is high staff turnover, the common attitude of team members depends upon project pressures, communication among team members, potential problems with compensation as well as benefits. Besides monitoring these factors, a project manager has also responsibility to monitor the efficiency of risk mitigation steps.

For example, a risk mitigation step which is pointed here is called for the definition of work product standards and mechanism to assure that there is no time delay in development of work product. It is responsibility of the project manager to monitor all the respective work products cautiously to assure that each can stand on its own and that each conveys data that would be essential if a newcomer were forced to join the existing software team in their work in the middle of the project.

Risk management and contingency planning considers that there is failure in mitigation efforts and that the risk is really occurred. Continuing the example, the execution of project is going smoothly and some team members announce that they will be leaving. If there is implementation of mitigation strategy, backup is on hand, information is well-documented, and knowledge has been dispersed across the team. Additionally, one can for the time being reform resources (and manage the project schedule again) to such functions which are completely staffed, enabling newcomer whose introduction in the team is necessary to "get up to speed."

It is essential to understand that risk mitigation, monitoring, and management (RMMM) steps will definitely incur additional project cost.

Q. 3 Explain change control activities in SCM. **Dec. 15, May 17, May 18**

Ans. :

Change Control (Configuration Control)

Changes occur at all phases in the software lifecycle. Design or implementation changes may be necessary if requirements change, or when deficiencies are identified in the design or implementation approach to a stable requirement. Testing may uncover defects that require changes in the code or the design and requirements. Changes must be made to the right version of the code, testing must verify performance of the change and the integrity of the remaining software, and all associated documentation must be updated to be consistent with the final code.

A mechanism is needed to process change requests (e.g., discrepancies, failure reports, requests for new features) from a variety of sources throughout the development process and during operation and support of the software product. This mechanism should extend to a definition of a process to track, evaluate, and implement change requests into a new version and new release of the software. Generally, no single procedure can meet the needs of all levels of change management and approval levels.

The minimum activities needed for processing changes include :

1. Defining the information needed for approving the requested change.
2. Identifying the review process and routing of information.

3. Describing the control of the libraries used to process the change.
4. Developing a procedure for implementing each change in the code, the documentation, and the released software product.

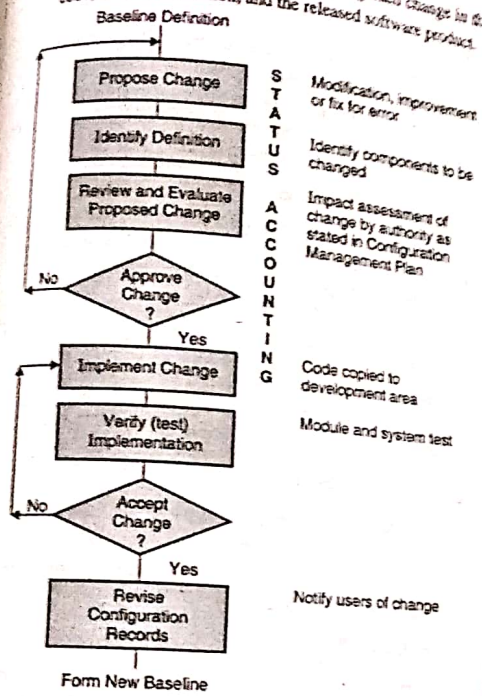


Fig. 5.2 : Configuration Control Process

Q. 4 Explain version control activities in SCM.

Dec. 15, May 17, May 18

Ans. : Version Control

Version control integrates the procedures with tools for the purpose of managing different versions regarding the configuration objects which are generated throughout the software process.

Following are the important capabilities which a version control system implements or is straightforwardly integrated with :

1. A project database (repository) which holds all the appropriate configuration objects,
2. A version management capability which holds all the relevant versions of a configuration object,
3. A make facility which helps to get all respective configuration objects and build a particular version of the software.

Additionally, version control as well as change control systems usually implement an issues/bugs tracking capability which helps the team to record and track the status regarding all of

the important issues related with each and every configuration object.

Most of the version control systems create a change set of all the respective changes which are necessary to build a particular version of the software. It is possible to identify various named change sets for an application or system. This helps in building a version of the software by mentioning the change sets (by name) which must be applied to the baseline configuration.

A system modeling approach is used to accomplish it. The system model contains :

(1) A template which contains a component hierarchy and a "build order" for those components which gives information regarding how the system must be constructed,
(2) Construction rules, and
(3) Verification rules.

Now-a-days there are several automated approaches have been proposed to version control. The basic distinction in those approaches is the sophistication of the attributes which helps in building specific versions.

Q. 5 Write note on Software Quality Management.

Ans. :

Software Quality Management

Software Quality Management is a method that guarantees that required level of software is achieved, so that the users are satisfied by its performance. The process involves quality assurance, quality planning, and quality control. This concept provides a complete understanding of Software Quality Management and illustrates the various steps involved in the process. Software Quality Management will make sure that the necessary level of quality is achieved by correcting the improvements to the product development process.

The main motive behind the SQM is to develop an ethnicity within the team and it is seen as everyone's liability. Software Quality management should be independent of project management to ensure the self-determination of cost and schedule adherences. Its effect directly impacts the process quality and indirectly affects the product quality.

Activities of Software Quality Management

1. **Quality Assurance**

QA aims at building measures and standards for maintaining quality at organizational level.

2. **Quality Planning**

To choose applicable procedures/actions and standards for a certain project, changes can be done as required to develop a quality plan.

3. **Quality Control**

It will make sure that most excellent practices and standards are followed by the software development team to maintain quality products. Quality of software will refer to software which is realistically error or defect free and which is delivered in time and within the allocated budget. It also meets the requirements and/or expectations, and is maintainable. In the software engineering term, software quality will indicate both the functional quality and structural quality. The different activities which are involved in the Software Quality Management are as follows :

Software Functional Quality

It indicates how well it satisfies a given function/design, based on the functional requirements or specifications.

Software Structural Quality

It handles the non-functional requirements that maintain the deliverance of the functional requirements, such as robustness or maintainability, and the extent to which the software was developed properly.

Software Quality Assurance

Software Quality Assurance (SQA) is a set of activities to ensure the quality in software engineering processes that finally results in quality software products. The activities establish and evaluate the processes that produce products.

Software Quality Control

Software Quality Control (SQC) is a set of activities to guarantee that the quality in software products should be maintained. These activities focus on determining the errors in the actual products produced. It deals with product-focused action. The quality of software has improved significantly over the past two decades. The reason behind is that companies are using new technologies for software development process such as object-oriented development, CASE tools and the testing technology such as Automation etc.

The software requirement should replicate the characteristics of the product that the client wants. The software quality attributes such as maintainability, usability, reliability cannot be accurately specified and measured in terms of software quality management. At the early stages of project it is very difficult to define a complete software specification. Therefore, it may happen that software conforms to its requirement, but the users don't get their quality expectations.

Software quality management is divided into three main categories :

(1) Quality assurance

The development of organizational procedures and standards that leads to high quality software.

(2) Quality planning

The selection of appropriate procedures and standards from project framework and used for a specific software project.

(3) Quality control

It defines the process ensuring that software development follows the quality procedures and standards. Quality management provides an independent test on the software and software development process. It ensures that project deliverables are consistent with organizational standards and goals.

Software Quality Challenge

In the software industry, the developers will never guarantees that the software is free of bugs, whereas industrial product manufacturers can give guarantees about their products. This difference is due to the following reasons.

(a) Product Complexity

Product complexity is the number of operational modes the product allows. Generally, an industrial product permits few modes of operations with different combinations of its machine settings.

(b) Product Visibility

Generally most of the defects in the industry can be detected during the manufacturing process.

Also the absence of a part in an industrial product can be easily detected in the product.

Process Quality Product

Quality of the developed products directly affects the quality of delivered products. For that reason the quality of the product can be calculated and the process is enhanced till the proper quality level is reached. Fig. 5.3 illustrates the process of quality assessment based on this approach. In manufacturing industry there is a clear connection among production process and product quality. However, quality of software is highly subjective to the skill of software engineers. In addition, it is hard to determine software quality attributes, like maintainability, reliability, usability, etc., and also to tell how process characteristics affect these attributes. However, from the past experience it has shown that process quality has a considerable demand on the quality of the software.

Process quality management have the following activities :

(1) Essential process standards.

(2) Monitoring the development process.

(3) Reporting the software.

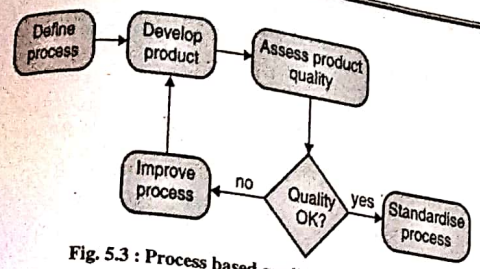


Fig. 5.3 : Process based quality assessment

Q. 6 Explain different metrics used for maintaining software quality.

Ans. : **May 15**

Metrics

Software metrics can be classified into three categories

1. **Product metrics**

It describes the character of the product such as volume, complexity, design features, performance, and quality level.

2. **Process metrics**

These characteristics can be used to progress the development and maintenance activities of the software.

3. **Project metrics**

This metrics describe the project characteristics and implementation. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity. Some metrics belong to many categories. For example, the in-process quality metrics of a project are both process metrics and project metrics. Software quality metrics is a division of software metrics that focus on the quality point of the product, process, and project. These are more strongly connected with process and product metrics than with project metrics.

Software quality metrics can be further divided into following categories :

1. **Mean Time to Failure**

It is the time difference between failures. This metric is generally used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

2. **Defect Density**

It deals with the errors relative to the software size articulated as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many business software systems.

3. **Customer Problems**

It measures the problems that customers encounter when using the product. It contains the customer's point of view towards the problem area of the software, which includes the non-defect oriented problems together with the defect problems.

4. **Client Satisfaction**

Client satisfaction is frequently measured by client survey data through the following five-point :

(i) Very satisfied (ii) Satisfied
(iii) Neutral (iv) Dissatisfied
(v) Very dissatisfied

Satisfaction with the quality of the product and its specific extent is usually gained through many methods of client surveys.

Based on the five-point data, many metrics with minor changes can be constructed and used.

For example

1. Percent of entirely satisfied clients
2. Percent of satisfied clients
3. Percent of dis-satisfied customers
4. Percent of non-satisfied customers

Usually, this percent satisfaction is used.

Q. 7 What is the philosophy of six sigma ? Explain six sigma strategies. **(8 Marks)**

Ans. :

Six Sigma

Six Sigma is a very much disciplined method which helps to focus on developing as well as delivering error free products and services.

Characteristics of Six Sigma

Six Sigma's aim is to deliver the product with time and with greater efficiency, thereby by increasing the customer satisfaction and by delivering what the customer is exactly expecting. Six Sigma follows a methodology which is structured, and it defines roles for the participants. Six Sigma is nothing but a data driven process, and it also requires correct data collection for the processes to be analyzed. Six Sigma is about to put the results on economical Statements.

Six Sigma approach is business-driven, multi-dimensional structured which includes :

Improving Processes

Lowering Defects

Reducing process variability

Reducing costs

Increasing customer satisfaction

Increased

Profits

The main aim behind Six Sigma is to improve the overall low quality defect rate in a process. It can be formally figure out from numerous data and can be shown by "zero defects" in practice and particularly that means 3.4 DPM (parts).

Key Concepts of Six Sigma

Six Sigma involves the a few key concepts.

1. **Significant to Quality**

   Attributes are most important to the customer.

2. **Defect**

   Failing to deliver what the customer exactly wants.

3. **Process Capability**

   What kind of process you can deliver ?

4. **Variation**

   What the customer sees and feels ?

5. **Stable Operations**

   Ensuring that the software should provide consistent, predictable processes to improve what the customer exactly wants.

6. **Design for Six Sigma**

   Designing to meet customer needs and process capability.

Mainly Six Sigma focuses on reducing process variation and then on upgrading the process capability.

Benefits of Six Sigma

Following are the benefits of Six Sigma :

1. Generates persistent success,
2. Sets a goal for everyone,
3. Improves value to customers,
4. Improves the rate of improvement,
5. Promotes learning.

Elements of Six Sigma

There are three main elements of Six Sigma :

1. **The Customers**

Customers mainly define quality. The customer always expect performance, consistency, competitive prices, prompt delivery, good service, clear and correct transaction and more. This means that it is very important to offer what the customers need to gain customer satisfaction.

2. **The Processes**

The central aspect of Six Sigma is to define the processes as well as their metrics and measures. In a software industry, the quality has to be always seen from the customer's viewpoint.

By understanding the lifecycle from the customer's perspective and processes, one can find out what the customer is seeing and feeling.

3. **The Employees**

For achieving most success all the employees of the Six Sigma process company can provide opportunities and incentives to employees to participate at their times and ability to lead, sustain and influence the company. The overall focus of Six Sigma is that all the best practices have proper roles and sensory-signals.

Responsibilities in a Six Sigma

There are some new responsibilities in a Six Sigma program, which are as follows :

1. **Leadership**

A leadership team always defines the goals and objectives of the Six Sigma process.

2. **Sponsor**

Six Sigma sponsors who are high-level individuals and they understand the Six Sigma processes and are committed to its success.

3. **Team Leader**

It is an individual responsible for managing the work of the team members and also acting as a mediator between the project sponsor and the team members.

4. **Black Belt**

The person having this belt has gained the highest skill level and also he is an experienced expert in various techniques. As per the Six Sigma program, the individual designated as a Black Belt has completed and gone through internal training activities and has the experience on several projects.

5. **Master Black Belt**

A person who deals with the team or its leadership, but is not a direct member of the team itself. This may be equivalent to the role played by the coach for more technical and complex projects. At the conclusion of the design phase, you should know who the customers or end-users are, their resistance issues, and requirements. You should also have a clear understanding of goals and the scope of the project including budget, time constraints, and deadlines.

Q. 8 What is FTR in SQA? What are its objectives? Explain steps in FTR.

Dec. 15, May 16, May 17, May 18

Ans. :

Formal Technical Review (FTR)

A formal technical review is a process performed to give assurance of software quality. This testing activity is done by software engineers and other persons.

The objectives of the Formal Technical Review are :

(1) To find out errors in function, logic, or coding in the software.

(2) To check that the software fulfils the requirements for which it is built.

(3) To give assurance that software has been designed as per customer requirements.

(4) To create software developed in uniform order.

(5) To create more manageable project.

The Formal Technical Review is used while providing training and it is also used as an example for junior developers to study different ways for software analysis, design, and development. The Formal Technical Reviews is a class of reviews that contains walkthroughs, inspections, round-robin reviews and other small technical assessments of software. Every Formal Technical Review is performed as a meeting. If Formal Technical Reviews is planned, controlled, and attended in correct way then only it becomes successful.

Steps In FTR

The review meeting

Every review meeting should be conducted by considering the following constraints :

1. **Involvement of people :** Between 3 to 5 people should be involve in the review.
2. **Advance preparation :** Advance preparation should occur but it should be very short. At the most 2 hours of work for each person can be spent in this preparation
3. **Short duration :** The duration of the review meeting should be less than two hours.

Rather than attempting to review the entire design, walkthrough are conducted for modules or for small group of modules. The focus of the FTR is on work product (a software component to be reviewed). The review meeting is attended by the review leader, all reviewers and the producer. The review leader is responsible for evaluating the product for its deadlines. The copies of product material are then distributed to reviewers. The producer organizes "walkthrough" for the product, explaining the material, while the reviewers raise the issues based on their advance preparation.

Review Reporting and Record Keeping

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting, and a review issues list is produced. In addition, a formal technical review summary report is completed. A review summary report answers three questions :

1. What was reviewed?
2. Who reviewed it?

3. What were the findings and conclusions?

The review summary report is a single page form (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties.

The review issues list serves two purposes :

(1) Identify problem areas within the product and

(2) Serve as an action item checklist that guides the producer as corrections are made.

An issues list is normally attached to the summary report. establish a follow-up procedure to ensure that items on the issues list have been properly corrected. Unless this is done, it is possible that issues raised can "fall between the cracks." One approach is to assign the responsibility for follow-up to the review leader.

Q. 9 Explain Walkthrough. **May 16**

Ans. :

Walkthrough

Walkthrough is a Review meeting but it is different from Inspection as it is not a formal process. Generally it is started by the author of code. In walkthrough, document or code is read by author and others can write note on the defects and suggestions about it. Walkthrough is informal way of testing so there is no need of moderator while performing walkthrough. We can call it as Open Ended discussion because preparation before meeting and creation of a list of observation is not necessary. It is the informal way of review so it does not focus on documentation. Defect tracking is challenging task in walkthroughs.

The following are the objectives of Walkthrough :

1. Understand and learn the development of software product till date.
2. Detecting defects in the developed software product.
3. To explain information present in the document.
4. To verify and discuss about the validity of the proposed system.
5. Reporting the suggestions given by the others (other employees).

Q. 10 Compare FTR and Walkthrough. **May 16**

Ans. :

Comparison between FTR and Walkthrough

Parameter	FTR	Walkthrough
Concept	A formal technical review is a process performed to give assurance of software	Walkthrough is a Review meeting but it is different from Inspection as it is not a

Parameter	FTR	Walkthrough
	quality	formal process
Performer	This testing activity is done by software engineers and other persons	Generally it is started by the author of the code
Process	In review, a work product is inspected for defects by individuals other than the person who produced it	In a Walkthrough, the producer describes the product and asks for comments from the participants
Work Product	A Work Product is any significant deliverable developed through the requirements, design, coding, or testing phase of software development	Walkthroughs generally help to examine source code as opposed to design and requirements documents. A step-by-step, line-by-line simulation of the code is done by the participants.
Way to perform	It is usually performed as a peer review without management involvement.	A walkthrough is particularly helpful for higher-level documents such as requirement specifications and architectural documents.

Chapter 6 : Software Testing and Maintenance

Q. 1 What are the objectives of testing ? **Dec. 16**

Ans. : Objectives of Software Testing

Finding defects which can be generated by the programmer while doing coding of software. Report the detected defects to developer for correction and after correction retest the software product to give assurance that software does not contain any defect. Software testing is done to give assurance that our software product fulfills all requirements of end user.

To give assurance that we deliver quality software to our end user.

1. To make sure that it satisfies requirements present in the BRS (Business Requirement Specification) and SRS (System Requirement Specifications).
2. To get the confidence of the customers by giving them a quality software product which must be user friendly and fulfills all the requirements and expectations of customer.

Q. 2 Explain the software testing process in detail.

Ans. :

Software Testing Process

Software testing process is a sequence of various activities which helps to certify software system. Software testing process activities are : Requirement analysis, Test planning, Test case development, environment set up, Test execution, Test cycle closures. Each of the activity of software testing process has defined entry and exit criteria.

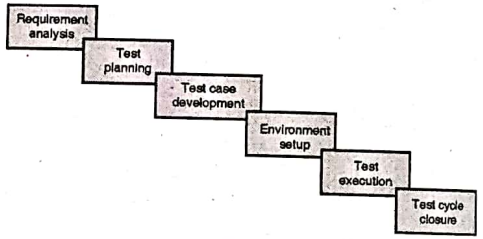


Fig. 6.1

1. **Entry criteria :** Entry criterions of testing are prerequisite conditions that must be satisfied before testing begins.
2. **Exit criteria :** An Exit criterion of testing describes the conditions that must be satisfied before testing is concluded.

Phases of Testing process

(i) Requirement Analysis

In this phase all gathered requirements of system (i.e. functional as well as non-functional) are studies to determine whether these requirements are testable or not. Depending on requirements, it is decided to perform type of testing, what should be focus and priority of testing. At the end of this phase Requirement Traceability Matrix (RTM) is prepared.

(ii) Test planning

It is also known as test strategy phase. It is responsibility of test manger. In this phase manager finds out required efforts and estimated cost for the system under development. At the end test plan is prepared as an outcome of test planning.

(iii) Test Case Development

In this phase all test data is determined, reviewed and rewritten if necessary. It includes generation, verification and test scripts are outcome of this phase.

(iv) Test environment set up

After completion of set up, smoke test is performed on it. This ready test environment and result of smoke test are output of this phase.

(v) Test Execution

During this phase testing of software is carried out by test team using previously prepared test plan and test cases. Intention of test execution is to find bugs in software. These bugs are reported to development team for correcting them. After correction test is performed in order to assure that reported bugs are removed.

(vi) Test cycle closure

This is end of software testing process. In this phase cycle completion criteria is determined depending on time, test coverage, quality of system, cost of software, business objectives etc. By using all these attributes test metric is prepared. Before this phase testing has been completed and defect logs are available. Using or analyzing defect logs, test closure report is prepared.

Q. 3 Write note on Strategic Approach to software testing.

Ans. :

Strategic Approach to Software Testing

Testing is a considered as a set of activities which can be planned in advance and implemented systematically. Hence a there is need to define template for software testing which will be a series of steps in which specific test case design techniques as well as testing methods can be placed.

There are number of software testing strategies available from which following characteristics are expected :

For effective testing, there is need of effective technical reviews. This helps to eliminate errors before testing commences. Testing begins with component level and proceeds "outward" toward the incorporation of the whole computer-based system. Different testing techniques are considered as suitable for different software engineering approaches and at different points in time. In case of small projects software developer conducts the testing but for large projects, there is need of an independent test group.

Testing and debugging are considered as two distinct activities, but it is necessary to accommodate debugging in any testing strategy. For a strategy of software testing it is important to accommodate low-level tests which are required to verify that implementation of a small source code segment as well as high-level tests which are used to validate most important system functions against customer requirements is correct. A strategy must offer guidance for the practitioner (newcomer) and a set of milestones for the well experienced manager. Since the execution of steps regarding the test strategy is done at a time when there is increase in deadline pressure, progress must be measurable and problem should be identified as early as possible.

Q. 4 Compare Verification and Validation Testing.

Ans. :

May 15, Dec. 17

Difference between Verification and Validation

Parameter	Verification	Validation
Focus	Concentrates on right way to build a system.	Concentrates on output of build system.
Definition	Verification can be defined as it is a process of estimating the intermediary work products of a software development lifecycle to verify that we are in the correct track of creating the end user product.	Validation can be defined as it is the process of evaluating the final product to ensure that the software meets all the specified requirements.
Aim	The aim of Verification is to ensure that the product being develop is as per the requirements and design specifications.	The aim of Validation is to ensure that the product really meets up the user's requirements, and verify whether the specifications be correct in the first place.
Inclusion	Verification contains Reviews, Meetings and Inspections.	Validation contains testing such as black box testing, white box testing, gray box testing etc.
Performer	Verification is	Validation is conducted

Parameter	Verification	Validation
	conducted by quality assurance team.	by testing team.
Execution of code	Execution of code is not a part of Verification.	Execution of code is a part of Validation.
Explanation	Verification process gives explanation about whether the outputs are as per the inputs or not.	Validation process gives explanation about acceptance of software by the user or not.
Processes	Plans, Requirement Specifications, Design Specifications, Code, and Test Cases etc. are evaluated in verification.	Testing of actual software is done in validation.

Q. 5 Explain unit testing. **Dec. 15**

Ans. :

Unit Testing

Software testing levels are the different stages of the software development lifecycle where testing is conducted.

There are four levels of software testing :

Unit >> Integration >> System >> Acceptance.

Unit Testing

Unit Testing is a level of software testing where individual units / components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc.

In object-oriented programming, the smallest unit is a method, which may belong to a base / super class, abstract class or derived/ child class. (Some treat a module of an application as a unit.

Unit Testing Method

It is performed by using the White Box Testing method.

When is it performed?

Unit Testing is the first level of software testing and is performed prior to Integration Testing.

Who performs it?

It is normally performed by software developers themselves or their peers. In rare cases, it may also be performed by independent software testers.

Unit Testing Tasks / Aspects of the software tested in unit testing

1.	Unit Test Plan	2.	Prepare
3.	Review	4.	Rework
5.	Baseline	6.	Unit Test Cases/Scripts
7.	Prepare	8.	Review
9.	Rework	10.	Baseline
11.	Unit Test	12.	Perform

Unit Testing Benefits

Unit testing increases confidence in changing / maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less. Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.

Development is faster. If there is no unit testing in place, developer writes the code and performs that fuzzy 'developer test' (Developer set some breakpoints, fire up the GUI, provides a few inputs that hopefully hit the code and hope that are all is set.) But, if there is unit testing in place, developer/tester writes the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests; there is no need to fire up the GUI and provide all those inputs. And, also, unit tests are more reliable than 'developer tests'.

Development is faster in the long run too. The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing. The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be scanned.

Q. 6 Enlist various types of errors detected by unit testing.

Ans. :

List of Errors Detected by Unit Testing

Following is the list of errors which are usually Unit Testing can detect :

1. Local data structure
2. Boundary conditions
3. Independent path
4. Error handling path
5. Local and global variable

6. Incorrect initialization
7. Incorrect symbolic representation of an expression
8. Incorrect arithmetic precedence
9. Global variable naming convention
10. Mapping error
11. Log and exception handling
12. Referred different data type in the code and database
13. Necessary input parameter not trim during the saving in the database.

Q. 7 Explain integration testing. **Dec. 15, Dec. 16**

Ans. :

Integration Testing

Integration testing is a method of software testing in which all units of software under test are integrated and tested as a single group. It is always done after unit testing is applied. It mainly focuses on testing data communication among all units of system. It makes use of mythologies as Big bang approach and incremental approach. Incremental approach can be either top-down, bottom-up or combination of both. The main objective of integration testing is to determine faults in communication between, integrated units.

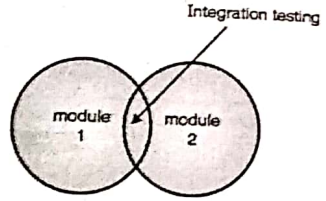


Fig. 6.2

Following are reasons for performing Integration testing :

1. A Module is developed by software developer whose level of understanding and programming logic may not be same as of other software developers from project team.
2. So, while performing Integration Testing, we required to check whether software module works efficiently with other modules or not.
3. While developing single module, there is risk of changes in requirements by the clients.
4. We may not perform unit testing for this new requirements and so performing integration testing of software is needed.
5. There is possibility that interfaces (through which operations on database are done) present between software modules and database may contain error.
6. There is possibility of error occurrence due to external hardware interfaces.

7. There is possibility of missing issues if exceptions are not handled properly.
8. For all these reasons we need to perform integration testing.

Types of Integration Testing

1. Big Bang Approach
2. Incremental Approach: which is categorized into given types :
   (a) Top Down Approach
   (b) Bottom Up Approach
   (c) Sandwich Approach - Combination of Top Down and Bottom Up

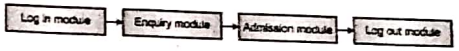
Q. 8 Write short note on System Testing. **May 17**

Ans. :

System Testing

System Testing is testing of entire and completely integrated software product. System Testing is sequence of various tests whose work is to test the complete computer based system. System testing is end-to-end testing. i.e. we test system from log in module to log out module.

For e.g. In college admission project, system testing carried out as



System testing contains both functional (check whether requirements of user are fulfilled or not) as well as Non-Functional testing (check whether expectations of user are fulfilled or not).

There are Two basic categories of Software Testing such as

1. **Black Box Testing**

   In this type of testing, we check working of our software project by running it. We do not need to check code of our software in this testing type. **System testing** is included in black box testing

2. **White Box Testing**

   In white box testing we check internal working of our code. So in this type of testing tester need deep knowledge of programming language which is used in our software product.

Types of System Testing

System Testing have more than 50 types.

1. **Usability Testing**

Usability Testing is a type of software testing in which a group of end-users of a software system use software to check user friendliness. It is non-functional testing. This testing basically concentrates on how easily user handle the application.

It is verified that after few hours training, end user can use system comfortably. Icon used in user interface is not confusing. e.g. for cut option we use scissor picture, not CD picture. This type of testing is also known as User Experience Testing. This testing is suggested to perform at the initial design phase of SDLC because it find out bugs (errors) in user interface and give chance to improve them.

2. Load Testing

Load testing is a type of Performance Testing which specifies performance of system under real-life load conditions. Load testing helps to specify how the application behaves when multiple users access it at same time. This testing normally used to find out :

1. The maximum number of users who can access the software at same time.
2. Specifies whether currently available infrastructure i.e. software and hardware is adequate to execute the application.
3. Check what happen when maximum number of users accesses the system simultaneously.
4. Scalability (action taken to increase capacity of system such as increase storage etc.) to permit to more number of users to access our application.

It is non-functional type testing.

Load testing is normally used when we test Client/Server based applications and Web based applications.

3. Regression Testing

Regression Testing is a type of software testing which is used to check whether changes have been made in code due to some error or change in requirement affects existing working functionality. In Regression Testing, we execute already executed test cases to give assurance that old functionalities work well after performing changes in code. This testing is performed to give guarantee that new code added in our software does not disturb working of existing functionalities.

4. Recovery Testing

Recovery Testing is done to specify whether system recovers itself after the system crash due to disaster such as power failure or network not available etc. In recovery testing, system perform rollback i.e. identify the point where the behavior of the system was correct and then from that point again perform the operations up to the point where system get crashed.

5. Migration Testing

Migration testing is performed to give assurance that the software can be moved from older system infrastructures to current system infrastructure without any problem. In Migration testing, we check data from old system for the compatibility with new system.

6. Functional Testing

Functional testing is a type of software testing which checks that every **function** present in the software application works as per requirements of user. This testing includes black box testing and it does not focus on the source code of the software application. Every functionality of the system is verified by tester using appropriate test data and actual result is compared with expected result. If there is difference between actual result and expected result then bug is considered. Functional testing is done using Requirement Specification Document.

7. Hardware/Software Testing

In Hardware/Software Testing, we perform testing of communication between the hardware and software used in our system. IBM called the Hardware/Software Testing as "HW/SW Testing".

Q. 9 Explain different techniques in white box testing. **Dec. 15, Dec. 17**

Ans. :

White box testing

Basis path testing is considered as a white-box testing technique.

It was proposed by Tom McCabe. These tests guarantee to execute every statement in the program at least once during testing. Basic set is the set of all the execution paths of a procedure.

Flow Graph Notation

Before basic path procedure is discussed, it is important to know the simple notation used for the representation of control flow. This notation is known as flow graph. Flow graph depicts control flow and uses the following constructs. These individual constructs are combined together to produce the flow graph for a particular procedure.

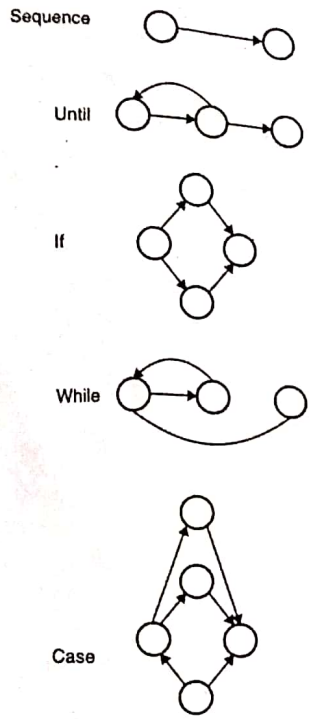


Fig. 6.3

Basic Terminology associated with The Flow Graph

1. **Node :** Each flow graph node represents one or more procedural statements. Each node that contains a condition is called a predicate node.
2. **Edge :** Edge is the connection between two nodes. The edges between nodes represent flow of control. An edge must terminate at a node, even if the node does not represent any useful procedural statements.
3. **Region :** A region in a flow graph is an area bounded by edges and nodes.

Q. 10 Explain cyclomatic complexity with suitable example. **Dec. 17, May 18**

Ans. :

Cyclomatic Complexity

Independent path is an execution flow from the start point to the end point. Since a procedure contains control statements, there are various execution paths depending upon decision taken on the control statement.

So Cyclomatic complexity provides the number of such execution independent paths. Thus it provides a upper bound for number of tests that must be produced because for each independent path, a test should be conducted to see if it is actually reaching the end point of the procedure or not.

Cyclomatic Complexity for a flow graph is computed in one of three ways :

1. The numbers of regions of the flow graph correspond to the Cyclomatic complexity.
2. 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 and

N is the number of flow graph nodes.

3. Cyclomatic complexity, V(G), for a graph flow G is also defined as

$$V(G) = P + 1$$

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

Example : Consider the following flow graph.

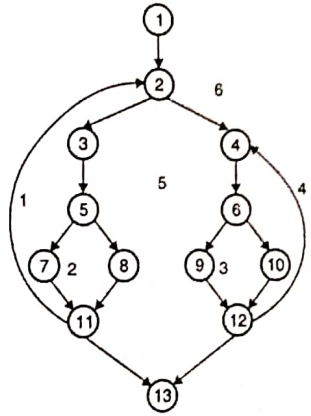


Fig. 6.4

Region, R = 6

Number of Nodes = 13

Number of edges = 17

Number of Predicate Nodes = 5

Cyclomatic Complexity, V(C)

$$V(C) = R = 6;$$

Or

$$V(C) = \text{Predicate Nodes} + 1$$
$$= 5 + 1 = 6$$

Or

$$V(G) = E - N + 2$$
$$= 17 - 13 + 2 = 6$$

Q. 11 Explain Black Box testing. **Dec. 10**

Ans. :

Black Box Testing

The software testing methods which test the functionality of an application without knowing its internal structure, coding information and knowledge of internal paths of the software is called black box testing. In this method test cases are built based on what application is supposed to do. It is also known as behavioural testing or specification based testing. In Black Box Testing we concentrate on inputs provided to software product and output from the software product without applying compulsion about knowledge of the programming language used in software under the test.

Black box testing can be applied during each level of software testing process. It makes use of various testing techniques like Boundary Value Analysis (BVA), equivalence class decision models etc.

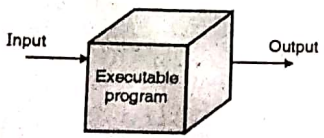


Fig. 6.5 : Black box testing

Following are the steps which are used to perform black box testing :

1. First the requirements and specifications of the system are analyzed.
2. Tester select valid test data i.e. data for positive test scenario to verify whether application under test is able to processes that data and give output as expected.
3. Also tester selects invalid test data for negative test scenario to verify whether application under test is able to processes that data and give output as expected.
4. Tester state expected outputs for each test data.
5. Software tester generates test cases with the selected test data. Then test cases are run.
6. Software tester performs comparison between the actual output and expected output.
7. Defect is detected if actual result is not same as expected result.
8. Then detected defect is reported to developer.
9. Developer can then fix that defect and then application is retested by tester.

For developing test cases following methods are used in Black Box Testing :

(i) Boundary Value Analysis (BVA)

As the name indicates boundary value analysis is a process of testing boundaries of the input values. It is most commonly used technique in BBT. Here basic idea is to select input values at their minimum, just above minimum, normal value, maximum value and just below maximum value. BVA always comes after the equivalence class partitioning.

(ii) Equivalence class partitioning

Equivalence class partitioning reduces the number of all possible inputs by dividing them into classes. It tests application thoroughly and avoids redundancy of input values. It can be applied at all levels of testing. It always performed before boundary value analysis technique is applied. BVA and equivalence partitioning are closely related and used together.

(iii) Cause - effect graphing

This technique graphically explains relationship between output of system and all the factors that affect the output of system. It helps to identify all possible causes of any specific effect. This technique tests only the external behaviour of system. It makes selection of test cases such that they logically relate cause to the effects.

(iv) Error - Guessing

Error guessing is a technique which makes use of tester's experience and skill for testing similar applications to find out defects which may not be determined by formal techniques. It is usually done after formal testing techniques are applied. It list outs all possible defects in system and then design tests which will attempt to produce them. The success of error guessing technique is mainly depends on knowledge and skills of tester.

Types of Black Box Testing

1. Functional testing

Functional testing is a type of software testing which checks that every function present in our software application works as per requirements of user or not. Functional testing includes black box testing and it does not focus on the source code of the software application. All functionalities of the system are verified by tester with the help of suitable test data and compare actual result with expected result. If there is difference between actual result and expected result then bug is detected. Functional testing is done using Requirement Specification Document.

2. Non-functional testing

In this type of black box testing, we do not perform testing to test specific functionality. Non-functional testing is used to check non-functional requirements such as performance, scalability, usability, and security are fulfilled or not.

3, Regression testing

Regression Testing is used to check whether changes which have been made in code because of some errors or change in requirement does not affect existing working functionality. In Regression Testing, we execute already executed test cases to give assurance that old functionalities work well after performing changes in code. This testing is performed to give guarantee that new code added in our software does not make any interruption in working of existing functionalities.

Advantages of Black Box Testing

(i) It is efficient for large systems.

(ii) It identifies contradictions in functional specification.

(iii) Detailed functional knowledge of system is not prerequisite for tester.

(iv) Tester and developer work independently.

Disadvantages of Black Box Testing

(i) It is difficult to find out all possible inputs for test-case in limited time.

(ii) Test method cannot be used for complex code.

(iii) Test cases cannot be designed without knowledge of functional specifications.

(iv) It may leave many program paths untested because of time constraint.

Q. 12 Compare white box testing and black box testing.

May 15, May 16

Ans. :

Difference between White Box Testing and Black Box Testing Technique

Parameter	Black box testing	White box testing
Applicable levels	It is mainly applicable at higher levels of testing : Acceptance and system testing.	It is mainly applicable at lower levels of testing i.e. unit and integration testing.
Base for testing	Requirement specification is the basis for Black Box Testing.	Detail design is the basis for White Box Testing.
Performed by	It is carried out by software testers.	It is carried out by software developers.
Aim	It aims at what functionality is system performing.	It aims at how system is performing its functionality.
Requirements	Programming knowledge and implementation knowledge is not necessary to carry out BBT.	Programming knowledge and implementation knowledge is necessary to carry out WBT.
Cost	It is less expensive than WBT.	It is more expressive than BBT
Focus on	In black box testing, we do not test the code and focuses on the functionality of software product.	In white box testing, we test the code and focuses on the security of software product and flow of control.
Facility provided	Black box testing provides facility of verifying communication among different modules present in software product.	White box testing does not provide facility of verifying communication among different modules present in software product.

Q. 13 Write short note on Software Maintenance.

May 17

Ans. :

Software Maintenance

Maintenance is required to give assurance that the software has ability to satisfy the changing requirements of user. Maintenance is important phase for software product which is developed using any SDLC model such as waterfall, spiral or iterative model etc. Due to corrective and non corrective software actions, we need to make changes in Software products.

Following are the needs of maintenance :

1. To correct the defects found in software product.
2. To improve the design of software.
3. To improve performance of software product.
4. To add new features.
5. To improve communication with other software.
6. To transfer legacy software system into new software system.
7. To replace the old software by new software.

Cost of Maintenance

Maintenance requires high cost. A most important part of the financial resources is spending on maintenance in a software life cycle.

A study on estimation of cost for software maintenance found that the cost of maintenance is as high as sixty seven percent of the cost of overall Software Development Lifecycle.

Making group of enhancements and corrections in management reports causes some misunderstanding related to high cost of corrections.

Understanding the types of software maintenance assist us to understand the structure of cost of software maintenance.

Understanding the factors that make bad impact on maintainability of software application can assists us to guess the cost of maintenance.

Few environmental aspects and their relationship to software maintenance costs includes following things :

1. Operating environment includes hardware and software.
2. Organizational environment includes policies, competition, process, product, and personnel.

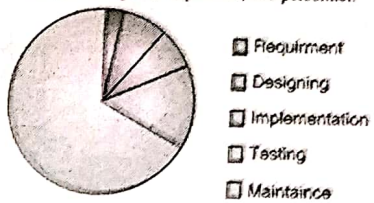


Fig. 6.6 : Cost of maintenance

The cost required for software maintenance is greater than fifty percent of all the phases included into the software development lifecycle.

There are number of factors, which increases cost of maintenance such as :

(a) Real-world factors which increase cost of Maintenance

The standard age of software application is assumed as up to 10 to 15 years. Legacy software which work on slow machines with less memory and storage capacity cannot prove itself better than newly coming enhanced software on modern hardware. As technology advances, it very difficult for maintenance of old software because the old software are costly. Most engineers who perform maintenance are new, inexperienced and use trial and error method to find out problems. Changes which may remain undocumented cause more problem in future.

(b) Software-end factors which increase Maintenance Cost

1. Structure of source code of software.
2. Programming Language.
3. External environment dependencies.
4. Staff availability and reliability.

Q. 14 What are the different types of maintenance?

May 16, Dec 16

Ans. :

Types of Software Maintenance

1. Corrective Maintenance

This type of maintenance contains changes and updations performed for fixing of defects found in software product that are detected by end user or tester. In corrective maintenance, we perform actions for correcting the defects in our software product that make bad effect on different parts of our software product such as design, logic or code. This type of maintenance is emergency maintenance in which unscheduled changes are done for temporarily keeping software in operation. Corrective maintenance has approach to examine the original specifications to state that what the system was originally designed to do.

Because of pressure given by management, the maintenance team release small codes for emergency corrections called as patching. Twenty percent part of all the maintenance activities are performed to do corrective maintenance.

2. Adaptive Maintenance

This category of maintenance contains making changes in software product to maintain the software product up-to date. Adaptive maintenance is used to make change in our software product according to the continually modifying world of technology and business environment. Adaptive maintenance is related with the modification in the application which is performed to make the application adaptable with ever changing environment.

Adaptive maintenance means applying changes to component of the software application which does not work properly due to change that is done in some other component of the software application. Adaptive maintenance includes updating software application according to modifications in the environment like the hardware or the operating system. Here, the concept environment is used to point to the conditions and the influences which are applied on the system.

For example, business rules, work patterns, and government policies are factors which affect the software application. If there is government policy that use 'European currency' only, will have a significant effect on the software system. To implement this modification, need those banks in different member countries to make changes in their software application to accept this currency. Adaptive maintenance contributes twenty five percent of all the maintenance activities.

3. Perfective Maintenance

Perfective Maintenance contains changes and updates performed to keep the software useful for long duration. These changes and updates contain new user requirements, new features for increasing the reliability and performance of software. Perfective maintenance is related with the modification in the software application which occurs because of addition of new functionalities into the application. Perfective maintenance basically associated with including new or modified user requirements.

Perfective maintenance includes performing enhancements in functionalities of application software in addition to enhance the performance of software application. It contains enhancement in both functionality and in efficiency of the source code and modifying the functionalities of the software application according to changing needs of users. Perfective maintenance contributes 50% of total maintenance which is the largest of all the maintenance activities.

4. Preventive Maintenance

Preventive Maintenance contains changes and updations to avoid problems in future software application. Its objective is to solve the problems which are not major at this point but may cause serious issues in future. Preventive maintenance includes applying changes to avoid the occurrence of errors. Preventive maintenance also includes doing actions to avoid the occurrence of errors. It tends to decrease the complexity of software application by improving understandability of program and growing maintainability of software application.

It includes documentation updating, code optimization, and code restructuring. Documentation updating includes making changes in the documents which are affected by the changes related to the present state of the system. Code optimization includes changing the programs to quickly execute the program or to use storage space in correct way. Code restructuring includes changing the structure of program for decreasing the complexity of source code and making it understandable. Preventive maintenance is performed only for maintenance organization and no outside requests are accepted for this category of maintenance. Preventive maintenance contributes 5% of all the maintenance activities.

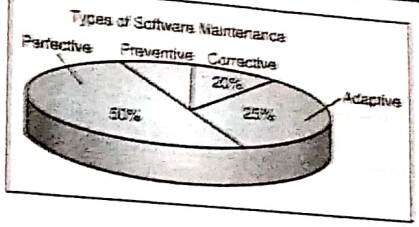


Fig. 6.7

Q. 13 Explain software reverse engineering in detail.

May 15, May 16, May 18

Ans. :

Reverse Engineering

It is a procedure to get system specification by analyzing and understanding the existing system. This procedure can be reverse software development life cycle model, i.e. go from maintenance phase to requirement gathering phase. Reverse engineering is the procedure that recognizes an object, a device, or technical properties of system through analysis of architecture of system, functions and operations. Consider an existing system has design about which we do not know anything. System designers perform reverse engineering by analyzing the source code and attempt to get the design. With the help of design, designer attempt to make conclusion about specifications of software product.

Reverse engineering is a process of analyzing an object to observe how software works in order to duplicate or add new features in the object. Idea of reverse engineering is invented in older industries and now many times it is used in computer hardware and software generation. Software reverse engineering includes reversing a machine code of program which is in the format of the sequence of 0s and 1s. It is transformed into the source code. Software reverse engineering is performed to find out the source code of a program since the source code need to study how the program do specific tasks, to enhance the performance of a program, to repair a bug i.e. correct an error detected in the program or to detect unsecure contents from program like virus.

Fig. 6.8 : Reverse Engineering

□□□