Module 1

Syllabus : Nature of Software, Software Engineering, Software Process, Capability Maturity Model (CMM), Generic Process Model, Prescriptive Process Models : The Waterfall Model, V-model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Agile process, Agility Principles, Extreme Programming (XP), Scrum, Kanban model.

Module 2

Module 3

Syllabus : Management Spectrum, 3Ps (people, product and process), Process and Project metrics.
Software Project Estimation : LOC, FP, Empirical Estimation Models : COCOMO II Model, Specialized Estimation Techniques.
Project scheduling : Defining a Task Set for the Software Project, Timeline charts, Tracking the Schedule, Earned Value Analysis.

Module 4 4-1 to 4-3

Chapter 4 : Software Design

Syllabus : Design Principles, Design Concepts, Effective Modular Design - Cohesion and Coupling, Architectural Design Component-level design, User Interface Design.

Chapter 5 : Software

Syllabus : Risk Identific process, Software Quali

Chapter 5 : Software Risk, Configuration Management & Quality Assurance 5-1 to 5-46

Syllabus : Risk Identification, Risk Assessment, Risk Projection, RMMM, Software Configuration management, SCM repositories, SCM process, Software Quality Assurance Task and Plan, Metrics, Software Reliability, Formal Technical Review (FTR), Walkthrough..

Module 6

6-1 to 6-45

ystem Testing,
ting, Software

CHAPTER 1

Module 1

Syllabus

Nature of Software, Software En
Prescriptive Process Models
Models, Concurrent Models, Ag

.1 Software

Q. 1.1.1 What is soft

Definition

Computer ***software****,*
instructions that tell
built, that actually p

– In computer
processed by
– Computer s
as online d
– Computer
own.
– At the l
individu
– A mach
change
– For ex
comp
– An i
syst
– Th
"ju

CHAPTER 1

Module 1

Introduction to Software Engineering and Process Models

Syllabus

Nature of Software, Software Engineering, Software Process, Capability Maturity Model (CMM). Generic Process Model, **Prescriptive Process Models :** The Waterfall Model, V-model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Agile process, Agility Principles, Extreme Programming (XP), Scrum, Kanban model.

1.1 Software

Q. 1.1.1 What is software? (Ref. sec. 1.1) **(2 Marks)**

Definition

*Computer **software**, or simply software, is a generic term that refers to a collection of data or computer instructions that tell the computer how to work, in contrast to the physical hardware from which the system is built, that actually performs the work.*

- In computer science and software engineering, computer software is; all the information processed by computer systems, programs and data.
- Computer software includes computer programs, libraries and related non-executable data, such as online documentation or digital media.
- Computer hardware and software require each other and neither can be realistically used on its own.
- At the lowest level, executable code consists of machine language instructions specific to an individual processor-typically a Central Processing Unit (CPU).
- A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state.
- For example, an instruction may change the value stored in a particular storage location in the computer - an effect that is not directly observable to the user.
- An instruction may also (indirectly) causes something to appear on a display of the computer system - a state change which should be visible to the user.
- The processor carries out the instructions in the order they are provided, unless it is instructed to "jump" to a different instruction, or is interrupted by the operating system.

- The majority of software is written in high-level programming languages that are easier and more efficient for programmers to use because they are closer than machine languages to natural languages.
- High-level languages are translated into machine language using a compiler or an interpreter or combination of both.
- Software may also be written in a low-level assembly language, which has strong correspondence to the computer's machine language instructions and is translated into machine language using an assembler.
- Fig. 1.1.1 shows how the user interacts with application software on a typical desktop computer. The application software layer interfaces with the operating system, which in turn communicates with the hardware. The arrows indicate information flow.

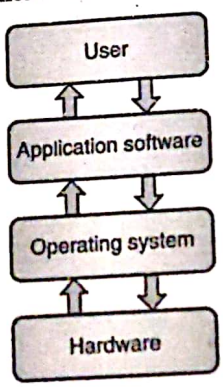


Fig. 1.1.1 : User interaction with application

Syllabus Topic : Nature of Software

1.2 Nature of Software

Q. 1.2.1 Explain changing nature of software. (Ref. sec. 1.2) (5 Marks)

- Now-a-days the software is not only remains a product, it also becomes an interface for delivering a product.
- As a product, software is able to deliver the computing potential which is basically embodied by computer hardware or precisely we can say, by a network of computers which one can access through local hardware.
- Software is considered as an information transformer or producer irrespective of whether it resides within a smart phone or functions inside a mainframe computer.
- Just like any transport vehicle, software can work as the basis for the control of the computer (operating systems), the transport of information (networks) and the generation and management of other programs (such as various software tools).

- Information delivery is the most important aspect of software. To make any data more useful in a local context, software can transform the data. It also helps in improving the competitiveness of a business by managing the business information.
- A gateway is provided by software to worldwide information networks (e.g., the Internet), and offer the way to gather information in several different forms.
- Over the last half-century, significant changes are occurred in the role of computer software.
- Now-a-days the software becomes more sophisticated as well as complex because of various aspects such as spectacular improvements in hardware performance, tremendous up-gradations in computing architectures, huge enhancements in memory and storage capacity, and extensive diversity of exotic input and output options.
- Sophistication as well as complexity leads to amazing results on success of systems, but they may also lead to huge problems for those who want to develop complex systems.
- In the economies of the industrialized world, the big software industry is considered as a dominant factor.
- In an earlier era, software was developed individually. Now-a-days an entire team works for it in which every person is responsible for one part of the technology which is necessary to deliver a complex application.

1.2.1 Defining Software

Q. 1.2.2 Define software. (Ref. sec. 1.2.1) **(2 Marks)**

Software can be defined in different ways :

Definitions

1. ***Software** is set of instructions (computer programs) that when executed provide desired features, function, and performance.*
2. ***Software** is data structures that enable the programs to adequately manipulate information.*
3. ***Software** is descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.*

- Now to understand the software exactly we have to examine some characteristics of it which make it different from remaining human made things.
- Software is considered as logical rather than a physical system element. Hence, there are characteristics of software which are significantly different than those of hardware.

1.2.1(A) Characteristics of Software

Q. 1.2.3 State any four attributes of good software. (Ref. sec. 1.2.1(A)) **(5 Marks)**

Q. 1.2.4 Describe the characteristics of software. (Ref. sec. 1.2.1(A)) **(5 Marks)**

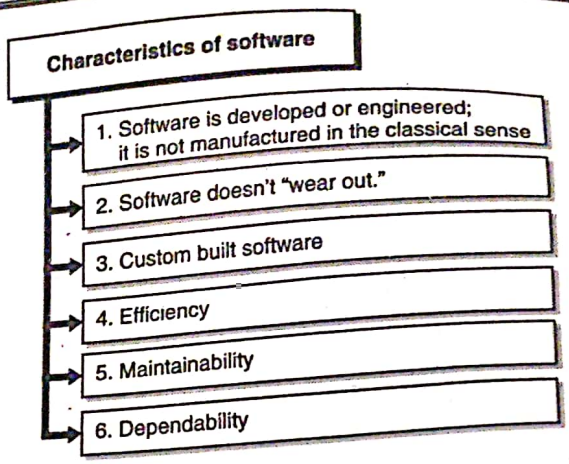


Fig. 1.2.1 : Characteristics of software

➜ **1. Software is developed or engineered; it is not manufactured in the classical sense**

- Even though there are some similarities found in software development and hardware manufacturing, both the activities are considered as fundamentally different.
- In both of the activities, good design is base for high quality, but in the process of manufacturing there may be quality problems in hardware which may not present in case of software.
- In both of the activities there is prominent dependency on people but there is vast difference in the relationship between people and work accomplished.
- The main aim of both the activities is construction of a "product", but the perspectives are not same.
- Software costs are concentrated in engineering which indicates that it is difficult to manage software projects similar to manufacturing projects.

➜ **2. Software doesn't "wear out"**

Q. 1.2.5 Elaborate the software characteristic "Software does not wear out". (Ref. sec. 1.2.1(A)(2)) **(5 Marks)**

- In case of hardware, the failure rates are high as compared to software early in its life. Such failures are mainly concerned with design or manufacturing defects. It is possible to correct the defects and drop the failure rate to a steady-state level for some time period.
- However after some time period, there is again increase in the failure rate as there are cumulative effects of dust, vibration, mishandling, tremendous high or low temperature, and number of other environmental maladies on components of hardware. In simple words, the hardware begins to *wear out*.
- Software does not have any risks of such environmental maladies which lead to wear out of hardware.
- In early life of software there may be high failure rates because of undiscovered defects. However, these can be corrected.

- Now it is clear that there is no possibility of software wear out but it does deteriorate.
- In it life change is an integral part of software. Whenever any change is made, there is possibility of introduction of errors which increase the failure rate.
- Before the software gets consistent state by corrections, any new change get introduced which again increases the failure rate.
- Gradually, the lowest failure rate level starts to increase which indicates that the software is deteriorating due to change.
- There is one important difference in hardware and software regarding wear aspect is that in case of hardware, a failed component can be replaced by spare parts. This facility is not available in software as there are no such spare parts.
- All the software failures point out that there is an error in design or in the method by which the respective design was transformed into machine executable code.
- Hence the tasks of software maintenance which handles requests for change has significantly more complexity as compared to hardware maintenance.

3. Custom built software

- Component reusability is an important aspect in software industry.
- It is responsibility of software engineer to design and implement a software component in such a way that it should be reused easily in many different programs.
- Latest reusable components summarize both data as well as the processing which is applied to the data, which helps the software engineer to develop new applications from existing components.
- For example, reusable components are used in the development of modern interactive user interfaces that enable the generation of graphics windows, pull-down menus, as well as wide range of interaction mechanisms.

4. Efficiency

Software is said to be efficient if it uses the available resources in the most efficient manner and produce desired result in timely manner.

5. Maintainability

- If customer requirements changes, programmers need to modify the software to fulfill those requirements.
- Software engineering provides the ability to maintain the software through modifying the software rather than changing whole product like hardware.

6. Dependability

- Dependability is the ability of the software that provides services which can be trusted by users.
- Dependability provides services to fulfill the customer's requirements, so that it is helpful to achieve trust of customers on the system.

1.2.2 Software Application Domain

Virtually on all computer platforms, software can be grouped into few broad categories.

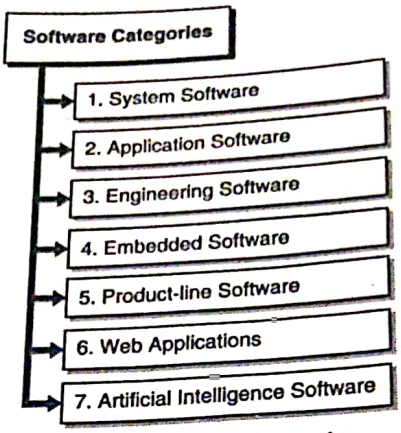


Fig. 1.2.2 : Software Categories

➜ **1. System software**

- This is a set of programs used to provide service to other programs. Few of them such as compilers, editors, and file management utilities are used to process complex but determinate information structures.
- Some other system software such as components of OS, several drivers, networking related software, and telecommunications processors are generally used to process heavily indeterminate data.
- In all the situations, the system software area is broadly depends upon the interaction with computer hardware.

➜ **2. Application software**

- These are the stand-alone programs which are specially developed for specific business needs.
- The Application Software helps to process business or technical data in such a way that it should be useful to manage business operations as well technical decision making.
- With traditional processing applications, one can also use application software to keep control on business functions in real time.

➜ **3. Engineering/scientific software**

- These types of software are characterized through the "number crunching" algorithms. There are number of engineering software in various fields like astronomy, space shuttle orbital dynamics, automotive stress analysis, automated manufacturing, molecular biology etc.
- However, there are drastic changes in modern applications within the engineering/scientific area as they are shifting from traditional numerical algorithms.

4. Embedded software

These are the software which are merged within electronic devices and are used for the purpose of implementing and controlling the features as well as functions for the end user as well as system itself.

5. Product-line software

- These software are developed to provide a particular functionality for use by several customers.
- The focus of product-line software may be on a restricted marketplace like inventory control applications or can focus on mass consumer markets such as word processing, spreadsheets, graphics based applications, multimedia, database management, and personal as well as business financial products.

6. Web applications

- These applications are also called as "WebApps". This is a network-centric software category which covers large number of applications.
- In shortest form, WebApps are same as of the group of linked hypertext files which provide information through text and limited graphics.
- Now-a-days the modern WebApps are evolving into environments of sophisticated computing which offer stand-alone features, computing functions, as well as content to the end user.
- It is also possible to integrate WebApps with corporate databases and business applications.

7. Artificial intelligence software

- In these applications non-numerical algorithms are generally used for the purpose of solving complex problems which cannot be handled by computation or straightforward analysis.
- There are various applications in this category such as robotics, games, expert systems, artificial neural networks, proving theorem, pattern matching like image and voice.

Syllabus Topic : Software Engineering

1.3 Software Engineering

(MU - Dec. 15)

Q. 1.3.1 Explain the term 'Software Engineering'. (Ref. sec. 1.3) (5 Marks)

Q. 1.3.2 Define 'Software Engineering'. (Ref. sec. 1.3) Dec. 15. 1 Mark

Software Engineering combines two concepts, Software and Engineering.

Software

- As we have already seen Software is set of executed programs related to specific functionality. Set of executable instructions is called as a program.

- 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. Software engineers design and build the software product.

☞ **Engineering**

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

☞ **Software Engineering**

☞ ***Definition 1***

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

OR

☞ ***Definition 2***

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.***

1.3.1 Layered Approach

Q. 1.3.3 Describe 4 layers of software engineering. (Ref. sec. 1.3.1) **(5 Marks)**

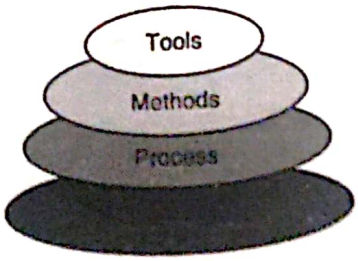


Fig. 1.3.1 : Layered Approach of software Technology

This approach is divided into 4 layers :

1. **A quality focus**
   - Any engineering approach must rest on the quality.
   - The most important aspect in software Engineering is Quality Focus.
2. **Process**
   - Foundation for SE is the Process Layer.
   - SE process is the GLUE that holds all the technology layers together and enables the tim development of computer software.
   - It forms the base for management control of software project.

3. **Methods**
   - SE methods provide the "Technical Questions" for building Software.
   - Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

4. **Tools**
   - SE tools provide automated or semi-automated support for the "Process" and the "Methods".
   - Tools are integrated so that information created by one tool can be used by another.

1.3.2 Activities of Software Engineering

Q. 1.3.4 Enlist Activities of Software Engineering. (Ref. sec. 1.3.2) (2 Marks)

Software Engineering includes many activities as follows :

- Understanding the user requirements is the first most important activity in software engineering.
- According to the requirements, the process of designing the system and taking decisions is implemented, which further leads to successful completion of development of product.
- Constructing the software product based upon designs and decisions made earlier is the next activity in software engineering.
- To verify the performance and quality of software, testing the software product is essential after it is constructed by software engineers.
- The next step is to provide maintenance for software product which is deployed to the customer.

1.3.3 Need of Software Engineering

Q. 1.3.5 Explain need of Software Engineering. (Ref. sec. 1.3.3) (2 Marks)

- As the technology changes, the user requirements and environment on which software is working also changes. So every organization is ranked based on the software engineering principles used by that organization.
- Implementing and managing large size of software, programmer requires a specific method to modularize the tasks so that size of software can't harm the software quality. Software engineering provides methodology for implementing complex software systems with high quality.
- Without any standard method or management, it is difficult to address defects in the product and correct them as early as possible. Software Engineering provides this functionality.
- Extending the previous software to add new functionality requires more cost in terms of time to develop and efforts taken by people, as compare to the process of developing new software to provide that functionality. Software engineering provides a way in which software system can be able to scale as needed in future.

Syllabus Topic : Software Process

1.4 Software Process

(MU - Dec. 15)

Q. 1.4.1 Explain in brief the software process framework. (Ref. sec. 1.4) Dec. 15. 4 Marks

– 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 selected process typically involves methods such as

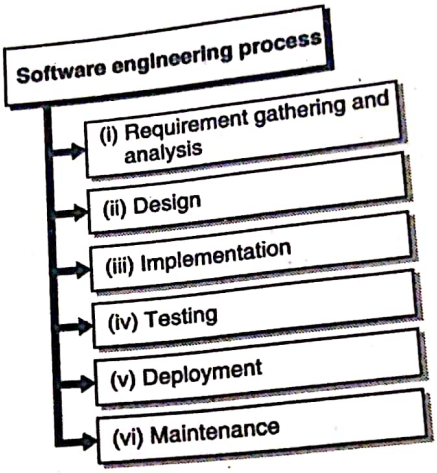


Fig. 1.4.1 : Software engineering process

The software life cycle follows the sequential flow in order to develop software.

We will see each and every stage of this life cycle in detail in subsequent points :

→ **(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 specificat document as an input.

– Based on requirement specifications software design is prepared.

– Output of this phase is design specification which specifies hardware and software requirem 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.

1.4.1 Comparison with Conventional Engineering Process

Q. 1.4.2 Compare Software engineering with Conventional Engineering Process. (5 Marks)

(Ref. sec. 1.4.1)

- Software engineering has some similar things as the principles of conventional engineering.
- It is assumed that software engineering is associated to the design of software or data processing products and belongs to its problem solving domain, including the class of problems related to software and data processing.
- This idea is expanded by drawing similarity from the methods that are generally used in engineering.
- It is observed that, just as the famous scientific method is used in the field of scientific research, the steps of engineering design process are used in the process of problem solving in the field of engineering.

- Following steps are mostly repetitive :
  1. Problem formulation,
  2. Problem analysis,
  3. Search for alternatives,
  4. Decision,
  5. Specification,
  6. Implementation
- It is advised that these steps are applicable to the field of software engineering as well.
- Software engineering is considered as a result of integration of hardware and systems engineering, including a set of 3 key elements - methods, tools and procedures which facilitate the manager to control the process of software development.
- The methods give the technical "how to's " for building software; tools offer automated or semi-automated support for methods; and procedures describe the series of applying methods, the deliverables, the controls, and the objectives.
- Compared to traditional engineering disciplines, software engineering demonstrates some remarkable differences.
- In conventional engineering, one moves from an abstract design to a concrete product. On other hand, in software engineering, one moves from design to coding.

Software Engineering	Abstract Design	→	More Abstract Code
Manufacturing Engineering	Abstract Design	→	Concrete Products

- The problem domains of software engineering can be anything, from word processing to real-time control and from games to robotics.
- In contrast to another engineering discipline, it is therefore much larger in scope and thus provides bigger challenges.
- Traditional manufacturing engineering that normally emphasize mass production is loaded with production features. Thus, it is highly production concentrated. Software engineering, in contrast, is inherently design concentrated.
- Product standardization assists in cost reduction in manufacturing, whereas such a possibility is remote in software engineering.
- The possibility of process standardization, however, is very high in the latter.
- Conventional engineering process makes the use of unlimited number of domain and application-specific ideas which are proved. Software engineering, in contrast, makes use of limited, but global concepts such as loops, conditions etc.

Syllabus Topic : Capability Maturity Model

1.5 Capability Maturity Model

Q. 1.5.1 Write note on Capability Maturity Model (CMM). (Ref. sec. 1.5) **(10 Marks)**

- 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 :

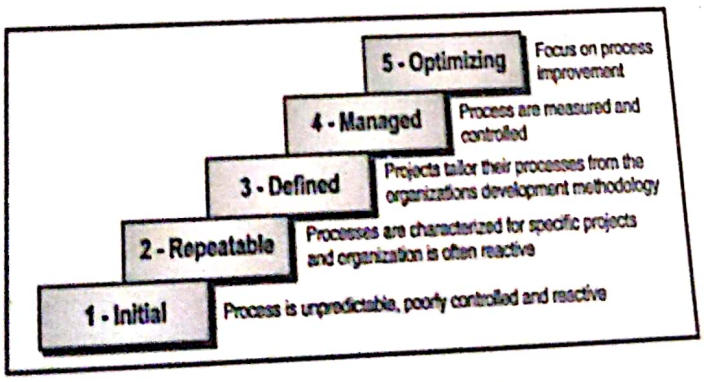


Fig. 1.5.1 : CMM Levels

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.

1.5.1 Capability Maturity Model Integration

Q. 1.5.2 Write note on Capability Maturity Model Integration (CMMI). (Ref. sec. 1.5.1) **(5 Marks)**

- The CMM model's application in software development has sometimes been problematic. Applying multiple models that are not integrated within and across an organization could be costly in training, appraisals, and improvement activities.
- The Capability Maturity Model Integration (CMMI) project was formed to sort out the problem of using multiple models for software development processes, thus the CMMI model has superseded the CMM model, though the CMM model continues to be a general theoretical process capability model used in the public domain.
- CMMI framework consists of a collection of computer programs based on knowledge, system engineering, software engineering, integrated product and process development and provider sourcing.

- CMMI framework has three groups as :
  1. CMMI for Development (CMMI - DEV)
  2. CMMI for Service (CMMI - SVC)
  3. CMMI for Acquisition (CMMI - ACQ)
- These three groups forms model components which are uniquely designed for particular business process and they may contain some core processes as a part of them which will be same among these groups.
- It is also possible to extend CMMI framework by making addition of model component to it.

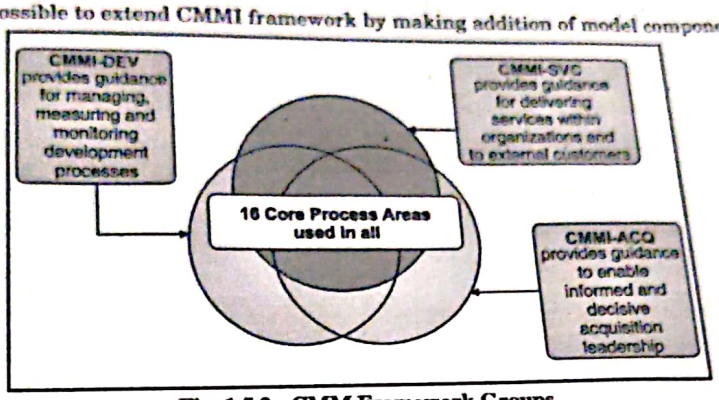


Fig. 1.5.2 : CMM Framework Groups

Syllabus Topic : Generic Process Model

1.6 Generic Process Model

Q. 1.6.1 Write note on Generic Process Model. (Ref. sec. 1.6) **(10 Marks)**

- In section 1.4 we have already seen the important phases of a software process which can also be termed as Software Development Life Cycle.
- Now we are going to discuss all these phases in detail for a generic process model with some important aspects.

System Engineering

- As software approximately always makes the modules of a larger system, one can starts work by forming requirements for all components of the system, recognizing not only the role performed by the software but, more importantly, its interface and interaction with the outside world.
- This 'system view' is necessary when software must interface with other elements such as hardware, people, databases and computers outside of, and ahead of the control of the system designer.
- In essence Systems engineering contains discovering some of the following issues :

  (i) Where does the software solution fit into the overall picture?

(ii) What does the system do? Is it necessary to examine some process or activity or is it simply performing analysis of data?

(iii) What environment will the system be placed in? That is system is on-line, real-time, embedded etc.

(iv) What user interface is required and how is it used?

(v) How systems communicate with other computers?

(vi) Does the system provide the required performance?

(vii) What time period has been planned and how serious it is. What budget does the customer have and is it realistic? What are the cost/benefits tradeoffs to the user in automating some manual procedure?

- When answers of these questions are received, the developer is in a good position to evaluate the RISK involved in implementing the system.

(1) Requirements Gathering and Analysis

- Requirements Analysis is the first important step towards creating a specification and a design.
- Trying to design a solution to a problem without totally understanding the nature and requirements of the user will always fails down the solution.
- It has been shown that more than ninety percent of software that are rejected by the customers after delivery is because the software does not meet the customer needs, expectation or requirements so it is important to understand those requirements completely.
- Requirements analysis is an iterative process carried out together by an analyst and the customer and represents an attempt, to discover the customer's needs, even as at the same time assessing the viability of a software solution.
- Analysis gives the designer the demonstration of the information which is processed by the system and the nature of the processing. i.e., what does the system do with the information which it processes?
- Analysis enables the designer to identify the software's function and performance. Also where the customer is not sure about how the system will eventually behave; the analyst may discover the concept of a prototype.

(2) Analysis Summary

The aim of requirements analysis is to generate a "requirements specification document" or a "target document" that illustrates great extent detail in an unambiguous a manner regarding exactly what the product should do. This requirements document will then form the basis for the succeeding design phase.

The requirements document may consists of :

- A Project plan containing details of delivery times and cost estimates.

- A model of the software's functionality and behavior in the form of 'data flow / UML diagrams' and, where suitable, any performance and data considerations, any input/output details etc.
- The results of any prototyping so that the emergence of the software and how it should behave can be exposed to the designer. This might contain a user's manual.
- Any formal Z or VDM specifications for the system requirements.
- Any test data that may be used to find out whether the end product is build according to the agreed specification or not.

The Fig. 1.6.1 illustrates the activities that are sum-up by the analysis.

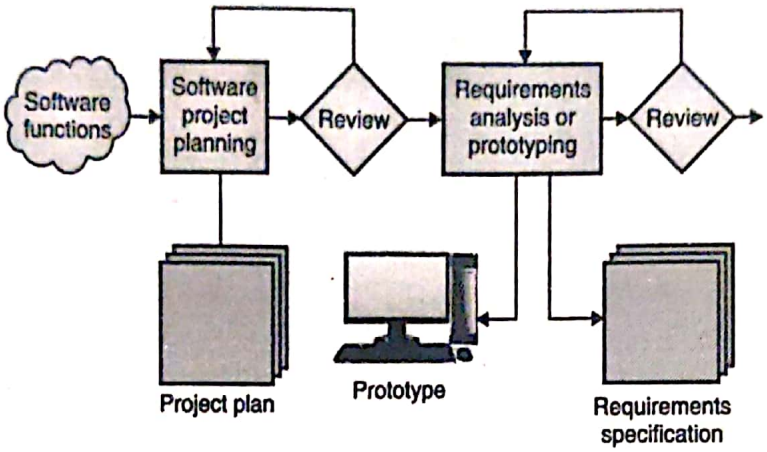


Fig. 1.6.1 : Activities in Analysis

(3) Design and Implementation (Coding)

When the analysis of the system has been completed, design or development can start. This is an attempt to convert a group of requirements and program/data models that are specified in the "requirements document" into a well-organized, designed and engineering software solution.

Design can be best summarized with the help of following steps :

- The data flow / UML (Unified Modeling Language) diagrams that signify the system model are transformed into appropriate hierarchical, modular program and data structure/architecture.
- Every program module is transformed into a correct cohesive function subroutine or class that is designed to do an individual task which is well-defined.
- Design further focus on the implementation of individual module/class. The user interfaces of module and its communication with other modules/objects is also considered and evaluated for good design.
- The algorithm of module can afterwards converted into a flowchart, which is a step-by-step graphical representation of the actions which are held by the module demonstrated in terms of sequence, selection and repetition.
- The flowchart can then be converted into Pseudo code, which conveys the same information as a flowchart, but presents it in a way that is more acceptable to translate into program code.

- At the end, the Pseudo code for every module is converted into a selected programming language and the number of modules entered are compiled and integrated into a system which is ready for testing.
- The final result of design and coding contains the architecture, data structures, flowcharts, Pseudo code, algorithms and all program source code listings, as depicted below.

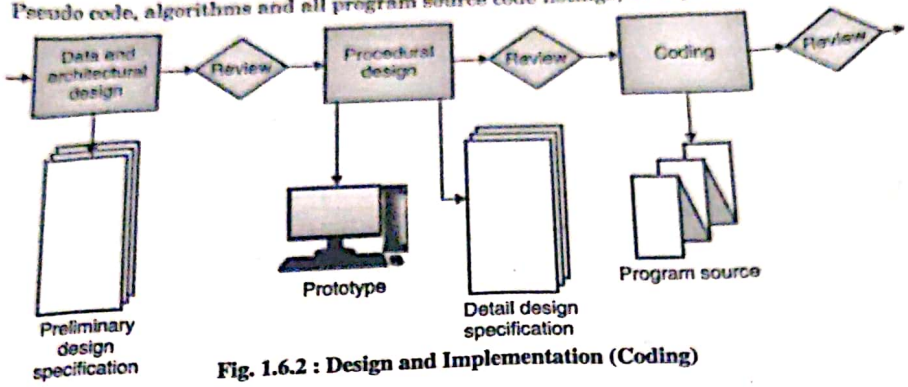


Fig. 1.6.2 : Design and Implementation (Coding)

(4) Software Testing

- When the part of the software components/modules has been written, testing and debugging can be started.
- Following techniques are involved in testing :
  - (i) **Verification and Validation :** This technique is useful for checking that the software meets the agreed specification and verifying that the software is correct in its operation.
  - (ii) **Black and white box testing techniques :** It is helpful for testing the insides of the modules for correct operation and testing the interfaces of the module.
  - (iii) **Integration Testing :** Testing that the modules all working together properly.
  - (iv) **Acceptance Testing :** Allowing the customer to test the product.
- Debugging can be defined as it is an art of recognizing the cause of failure in a part of software and correcting it.

(5) Deployment

After completion of software development, we have to install it at client site. This process is known as deployment.

(6) Software Maintenance

Software maintenance reapplies all of the previous life cycle steps to an existing program instead of a new one in order to correct existing or insert new functionality.

Syllabus Topic : Prescriptive Process Models

1.7 Prescriptive Process Models

Q. 1.7.1 Explain Prescriptive Process Models (Ref. sec. 1.7) **(5 Marks)**

- The following framework activities are performed irrespective of the process model selected by the organization.
  1. Communication
  2. Planning
  3. Modelings
  4. Construction
  5. Deployment
- The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

Syllabus Topic : Waterfall Model

1.7.1 Waterfall Model

Q. 1.7.2 What is waterfall model ? State the practical situations in which it can be used. (Ref. sec. 1.7.1) **(5 Marks)**

Q. 1.7.3 Explain the waterfall model. (Ref. sec. 1.7.1) **(5 Marks)**

- 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.

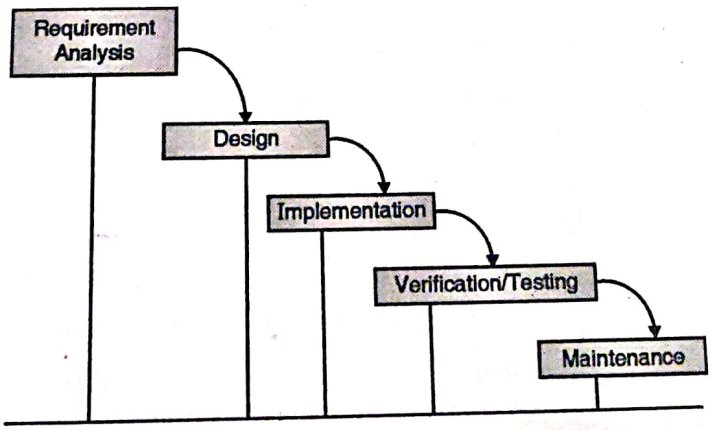


Fig. 1.7.1 : Waterfall Model

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***

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.

- Technology is understood.
- There are no ambiguous requirements.
- Ample resources with required expertise are available freely.
- The project is short.

Syllabus Topic : V-Model

1.7.2 V-Model

Q. 1.7.4 Write note on V-Model with verification and validation phases. (Ref. sec. 1.7.2) **(10 Marks)**

- In software development, the V-model represents a development process that may be considered an extension of the waterfall model, and is an example of the more general V-model.
- Instead of moving down in a linear way, the process steps are bent upwards after the coding phase to form the typical V shape.
- The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.
- The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction respectively.

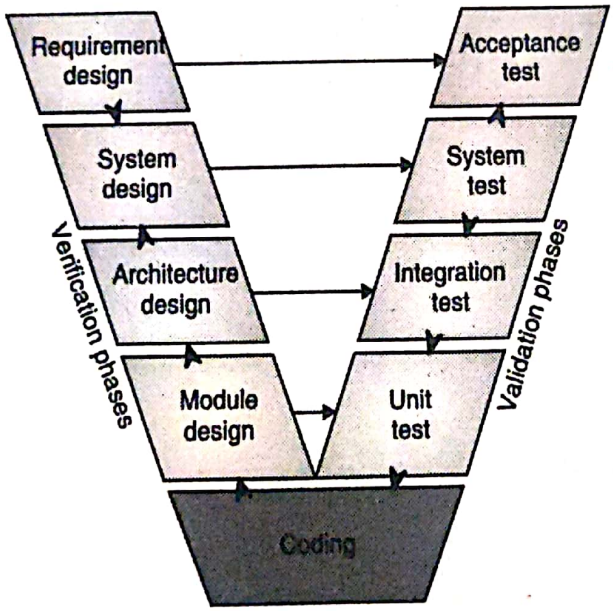


Fig. 1.7.2 : V-Model

- This model is basically divided into two phases; verification and validation.

1.7.2(A) Verification Phases

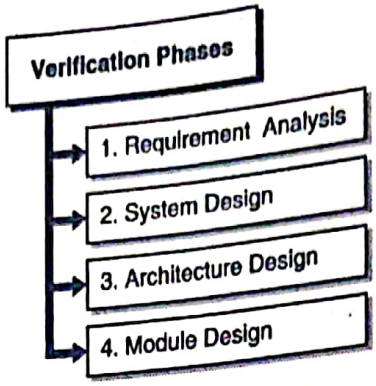


Fig. 1.7.3 : Verification phases

➜ **1. Requirements Analysis**

- In the requirements analysis phase, which is the first step in the verification process, the requirements of the system are collected by analyzing the needs of the user(s).
- This phase is concerned with establishing what the ideal system has to perform. However it does not determine how the software will be designed or built.
- Usually, the users are interviewed and a document called the user requirements document is generated.
- The user requirements document will typically describe the system's functional, interface, performance, data, security, etc. requirements as expected by the user.
- It is used by business analysts to communicate their understanding of the system to the users.
- The users carefully review this document as this document would serve as the guideline for the system designers in the system design phase.
- The user acceptance tests are designed in this phase.
- There are different methods for gathering requirements of both soft and hard methodologies including; interviews, questionnaires, document analysis, observation, throw-away prototypes, use case and static and dynamic views with users.

➜ **2. System Design**

- Systems design is the phase where system engineers analyze and understand the business of the proposed system by studying the user requirements document.
- They figure out possibilities and techniques by which the user requirements can be implemented.
- If any of the requirements are not feasible, the user is informed regarding the issue. A resolution is found and the user requirement document is edited accordingly.
- The software specification document which serves as a blueprint for the development phase is generated.
- This document contains the general system organization, menu structures, data structures etc. It may also hold business scenarios, sample windows, reports for the better understanding.

- Other technical documentation like entity diagrams, data dictionary will also be produced in this phase. The documents for system testing are prepared here.

➜ **3. Architecture Design**

- This phase of the design of computer architecture and software architecture can also be referred to as high-level design.
- The baseline in selecting the architecture is that it should realize all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology details etc.
- The integration testing design is carried out in the particular phase.

➜ **4. Module Design**

- The module design phase can also be referred to as low-level design. The designed system is divided into smaller units or modules and each of them is explained so that the programmer should be able to start coding directly.
- The low level design document or program specifications will contain a detailed functional logic of the module, in pseudo-code :
  - Database tables, with all elements, including their types and sizes.
  - All interface details with complete API (Application Programming Interface) references.
  - All dependency issues.
  - Error message listings.
  - Complete input and outputs for a module.
- The unit test design is developed in this stage.

1.7.2(B) Validation Phases

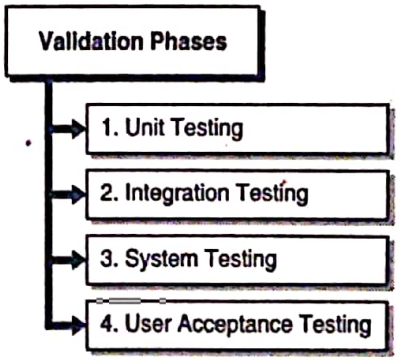


Fig. 1.7.4 : Validation phases

In the V-model, each stage of verification phase has a corresponding stage in the validation phase.

The following are the typical phases of validation in the V-Model :

1. Unit Testing

- In the V-Model, Unit Test Plans (UTPs) are developed during module design phase.
- These UTPs are executed to eliminate bugs at code level or unit level.
- A unit is the smallest entity which can independently exist, e.g. a program module.
- Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/units.

2. Integration Testing

- Integration Test Plans are developed during the Architectural Design Phase.
- These tests verify that units created and tested independently can coexist and communicate among themselves.
- Test results are shared with customer's team.

3. System Testing

- System Tests Plans are developed during System Design Phase.
- Unlike Unit and Integration Test Plans, System Test Plans are composed by client's business team.
- System Test ensures that expectations from application developed are met. The whole application is tested for its functionality, interdependency and communication.
- System Testing verifies that functional and non-functional requirements have been met.
- Load and performance testing, stress testing, regression testing, etc., are subsets of system testing.

4. User acceptance Testing

- User Acceptance Test (UAT) Plans are developed during the Requirements Analysis phase.
- Test Plans are composed by business users. UAT is performed in a user environment that resembles the production environment, using realistic data. UAT verifies that delivered system meets user's requirement and system is ready for use in real time.

Advantages of V-model

1. This model is simple and easy to use.
2. The testing activities such as planning, test designing are occurred prior to coding. This avoids wastage of time. Hence it has better chance of success compared to waterfall model.
3. Proactive defect tracking - that means the diagnosis of defects is done at early stage.
4. Avoids the downward flow of the defects.
5. It is considered as good for small projects in which requirements are easily understood.

Disadvantages of V-m

1. This model is con
2. The developmen regarding the so
3. If there are cha requirement doc

1.8 Incremental Pr

Q. 1.8.1 Discuss inc (Ref. sec.

- The increment
- The series of functionality
- After the first
- Based on cus made accordi
- This process The increme

Following tasks are

1. Communi
2. Planning functions a
3. Modellin
4. Construc
5. Deploym

Characteristics

1. System i
2. Partial s
3. First ta
4. The req

☞ **Disadvantages of V-model**

1. This model is considered as very rigid and least flexible.
2. The development of software is done in the implementation phase hence no early prototypes regarding the software are produced.
3. If there are changes in midway, then there is need to update the test documents along with requirement documents.

Syllabus Topic : Incremental Process Models

1.8 Incremental Process Models

➔ **(MU - Dec. 16)**

Q. 1.8.1 Discuss incremental model for software development with merits and demerits. (Ref. sec. 1.8) **Dec. 16, 5 Marks**

- 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.

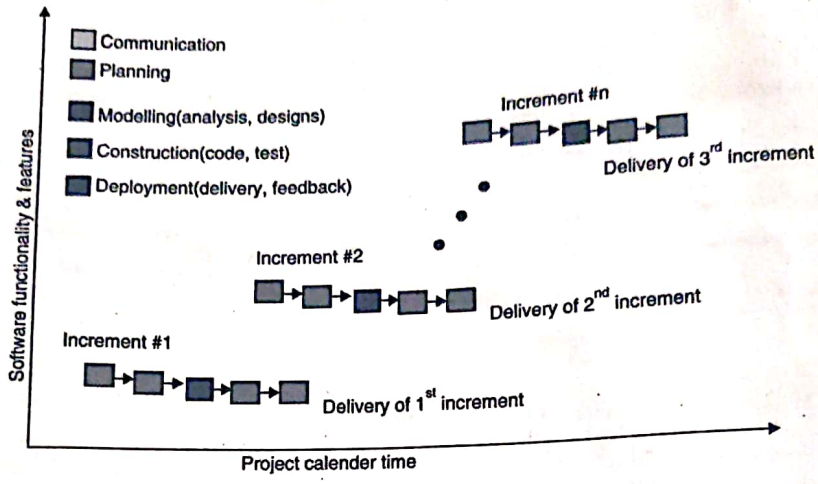


Fig. 1.8.1 : Incremental Model

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.

1.8.1 Difference between Waterfall Model and Incremental Model

Q. 1.8.2 Differentiate between waterfall model and incremental model. (Ref. sec. 1.8.1) **(5 Marks)**

Parameter	Waterfall Model	Incremental Model
Simplicity	Simple	Intermediate
Risk Involvement	High	Easily manageable

Parameter	Waterfall Model	Incremental Model
Flexibility to change	Difficult	Easy
User involvement	Only at beginning	Intermediate
Flexibility	Rigid	Less Flexible
Maintenance	Least	Promotes maintainability
Duration	Long	Very Long

Syllabus Topic : Evolutionary Process Models

1.9 Evolutionary Process Models

- Normally the Evolutionary Development Model is based on the principle that "stages consist of growing increments of an operational software product, with the way of evolution being discovered by operational experience".
- According to the business need and the changing nature of the market there are lot of improvements required in the software product over a time.
- Due to this lot of improvement is required in the product hence evolutionary developments model is iterative in nature.
- There are two types of models in Evolutionary process. They are :

  1. Spiral model 2. Prototyping model

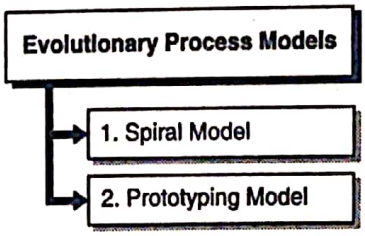


Fig. 1.9.1 : Evolutionary Process Models

1.9.1 Spiral Model

→ (MU - Dec. 17)

Q. 1.9.1 With a neat diagram explain the Spiral model of software development. (Ref. sec. 1.9.1) Dec. 17, 10 Marks

- 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.

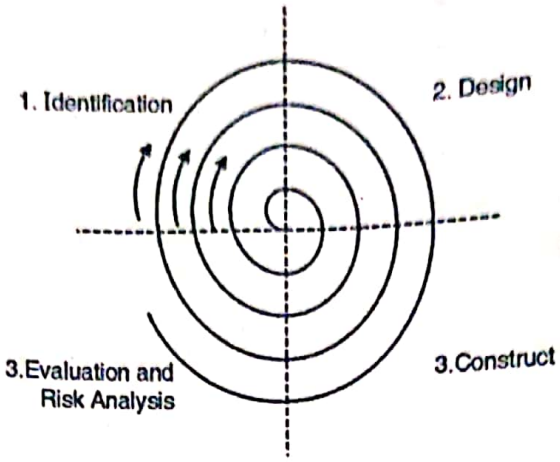


Fig. 1.9.2 : Spiral Model

(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.

1.9.2 Prototyping Model

→ (MU - Dec. 16)

Q. 1.9.2 Discuss prototyping model for software development with merits and demerits. (Ref. sec. 1.9.2) **Dec. 16, 5 Marks**

- 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.

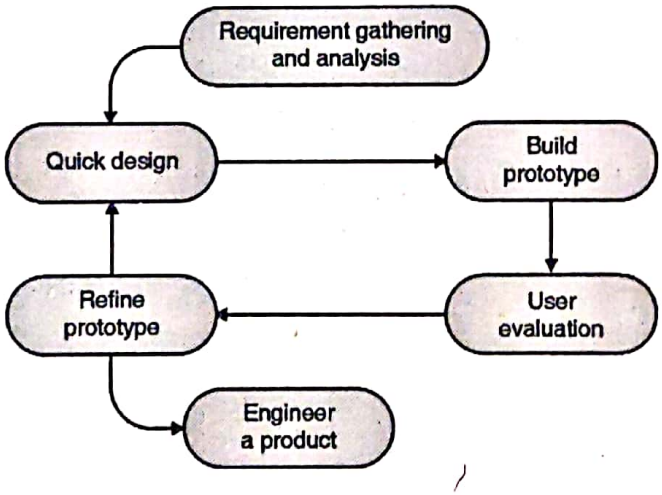


Fig. 1.9.3 : Prototyping Model

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.

When to use Prototype model

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically online system in which web interfaces have a very high amount of interaction with end users, are best suited for Prototype model.
- It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system.
- They are excellent for designing good human computer interface systems.

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

→ (MU - Dec. 17)

Q. 1.9.3 What are the potential problems of Prototyping Model ? (Ref. sec. 1.9.2) **Dec. 17. 5 Marks**

(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.

Syllabus Topic : Concurrent Models

1.10 Concurrent Models

Q. 1.10.1 Write note on Concurrent Models. (Ref. sec. 1.10) **(10 Marks)**

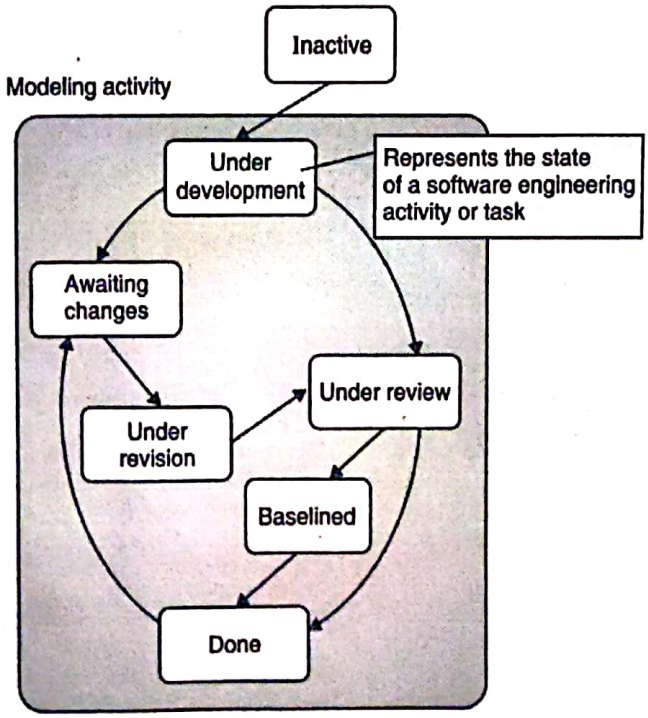


Fig. 1.10.1 : Concurrent Models

- The concurrent development model is also known as concurrent engineering.
- This model helps software team in the representation of iterative as well as concurrent elements of various process models.
- For example, consider the modeling activity which has been defined for the spiral model is carried out by calling one or more software engineering actions such as prototyping, analysis, and design.
- In Fig. 1.10.1 we can observe schematic representation regarding a software engineering activity in the scope of the modeling activity with the help of concurrent modeling approach.
- The state of activity **modeling** may be any out of the states which are noted at any particular time.
- In the same way, it is possible to represent remaining activities, actions, or tasks such as **communication, construction** etc. in an analogous manner.
- All the activities regarding software engineering are executed simultaneously but exist in different states.
- For example, consider in the early phase of a project, first iteration of the communication activity (not displayed in the diagram) has been completed and exists in the **awaiting changes** state.
- The modeling activity whose state is **inactive** at the time of completion of initial communication, now changes its state to **under development.**
- However, if the customer enforces that changes in requirements should be necessarily done, the modeling activity shifts from the **under development** state into the **awaiting changes** state.
- Concurrent modeling describes a sequence of events which will activate transitions in between states for all the software engineering activities, actions, or tasks.
- For example, in the previous phase of design (an important action regarding software engineering which occurs in the process of modeling activity), an inconsistency in the requirements model is uncovered.
- Because of it the event **analysis model correction,** is generated that will activate the requirements analysis action from the **done** state into the **awaiting changes** state.
- Concurrent modeling is proved to be compatible for all the types of software development and offers an exact picture of the ongoing state of a project.
- Instead of encircling all the software engineering actions, and tasks to a series of events, a process network is defined by the concurrent modeling.
- All the activities, actions, or tasks on the network execute concurrently with other activities, actions, or tasks.
- In the process network, events which are generated at one point activate transitions between the states.

Syllabus Topic : Agile Process

1.11 Agile Process

→ **(MU - May 15, Dec. 15, May 16, Dec. 16, May 17, May 18)**

Q. 1.11.1 What is Agile Methodology ? (Ref. sec. 1.11) **May 15, 4 Marks**

Q. 1.11.2	What is agility in context of software Engineering ? (Ref. sec. 1.11)	Dec. 15, May 17, 5 Marks
Q. 1.11.3	What are Agile Processes ? (Ref. sec. 1.11)	May 16, 5 Marks
Q. 1.11.4	Write short note on Agile Methodology. (Ref. sec. 1.11)	Dec. 16, 10 Marks
Q. 1.11.5	Write short note on Agile Process Models.(Ref. sec. 1.11)	May 18, 10 Marks

- 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:
  - Individuals and Interactions over processes and tools.
  - Working Software over comprehensive documentation.
  - Customer Collaboration over contract negotiation.
  - Responding to Change over following a plan.
- As per the view of Scott Ambler (Canadian software engineer, consultant and author) :
  - Tools and processes are important, but it is more important to have competent people working together effectively.
  - 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.
  - A contract is important but is no substitute for working closely with customers to discover what they need.
  - 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.

Syllabus Topic : Agility Principles

1.11.1 Agility Principles

→ (MU - May 15, Dec. 17)

Q. 1.11.6	Explain principles of Agile Methodology. (Ref. sec. 1.11.1)	May 15, 4 Marks
Q. 1.11.7	Describe and discuss characteristics of Agile Process Model. (Ref. sec. 1.11.1)	Dec. 17, 10 Marks

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.

1.11.2 Advantages of Agile Process

→ (MU - May 16)

Q. 1.11.8	What are Advantages of Agile Processes ? (Ref. sec. 1.11.2)	May 16, 5 Marks

There are number of advantages of Agile Process :

- Stakeholder Engagement,
- Transparency,
- Early and Predictable Delivery,
- Predictable Costs and Schedule,
- Allows for Change,
- Focuses on Business Value,
- Focuses on Users,
- Improves Quality.

1.11.3 Difference be

Q. 1.11.9 Differen (Ref. s

Parameter	P
Basic aim	Devel to the
Functionality	It requi
Popularity	It is
Examples	Wat

1.12 Extreme

Q. 1.12.1 E

– Extrem softwar

1.11.3 Difference between Prescriptive Process Model and Agile Process Model

Q. 1.11.9 Differentiate between prescriptive process model and agile process model (any four points). (Ref. sec. 1.11.3) **(5 Marks)**

Parameter	Prescriptive Process Model	Agile Process Model
Basic aim	Developed to bring order and structure to the software development process.	These models satisfy customer through early and continuous delivery.
Functionality	It can accommodate changing requirements	Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software.
Popularity	It is more popular.	Comparatively less popular.
Examples	Water fall model, Incremental models	Scrum, eXtreme Programming (XP), Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD).

Syllabus Topic : Extreme Programming (XP)

1.12 Extreme Programming (XP)

→ **(MU - Dec. 15, May 17)**

Q. 1.12.1 Explain Extreme Programming (XP) with suitable diagram. (Ref. sec. 1.12) **Dec. 15, May 17, 5 Marks**

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

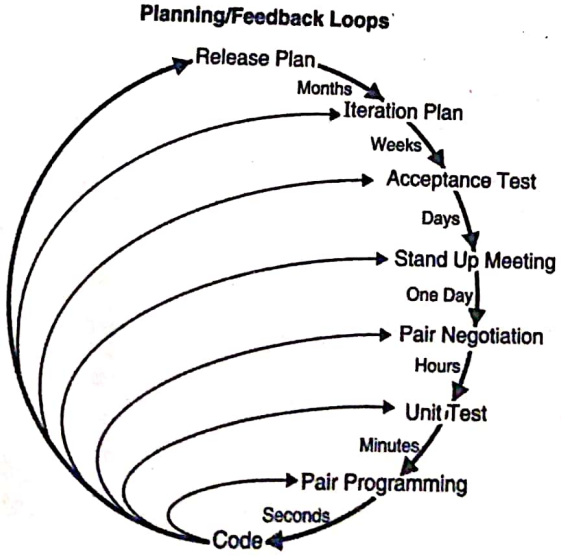


Fig. 1.12.1 : Extreme Programming

- 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.

1.12.1 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. Those five values are described below.

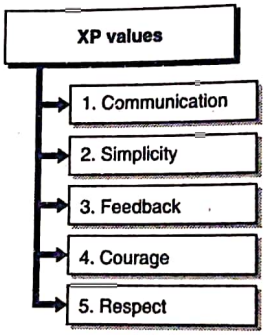


Fig. 1.12.2 : XP values

1. Communication

- Building software systems requires communicating system requirements to the developers of the system. In formal software development methodologies, this task is accomplished through documentation.
- Extreme programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge among members of a development team.
- 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 ca. 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.
- Proponents of XP acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; their claim is that this is more than compensated for by the advantage of not investing in possible future requirements that might change before they become relevant.
- Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed, while perhaps delaying crucial features.
- 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 :
  - **Feedback from the system :** By writing unit tests, or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.
  - **Feedback from the customer :** The functional tests (aka acceptance tests) are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
  - **Feedback from the team :** When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.
- 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.
- The direct feedback from the system tells programmers to recode this part.
- A customer is able to test the system periodically according to the functional requirements, known as user stories.
- As per Kent Beck (American software engineer and the creator of extreme programming), "Optimism is an occupational hazard of programming. Feedback is the treatment".

4. Courage

- Several practices represent courage. One is the commandment to always design and code for today and not for tomorrow.

- 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.
- Another example of courage is knowing when to throw code away: courage to remove source code that is obsolete, no matter how much effort was used to create that source code.
- 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.
- Adopting the four earlier values leads to respect gained from others in the team.
- Nobody on the team should feel unappreciated or ignored. This ensures a high level of motivation and encourages loyalty toward the team and toward the goal of the project.
- This value is dependent upon the other values, and is oriented toward teamwork.

Syllabus Topic : Scrum

1.13 Scrum

(MU - Dec. 17)

Q. 1.13.1 Write short note on SCRUM. (Ref. sec. 1.13) Dec. 17, 10 Marks

- 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.

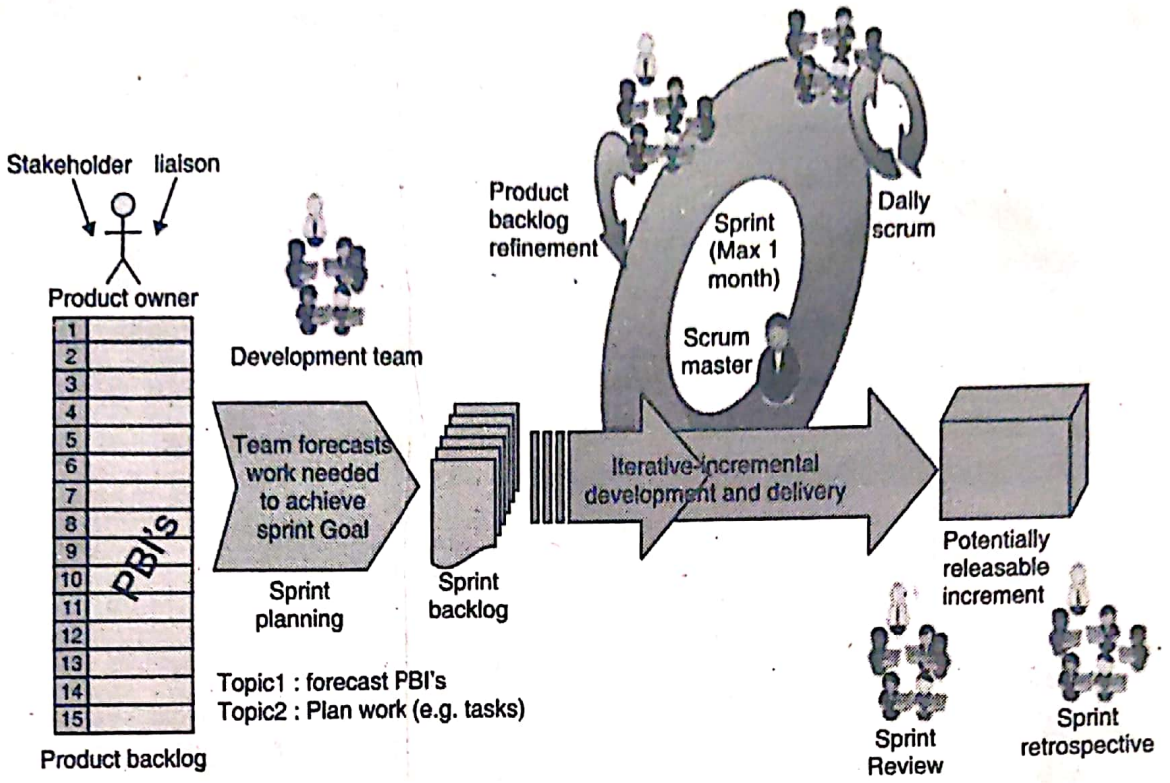


Fig. 1.13.1 : Scrum Framework

- 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.

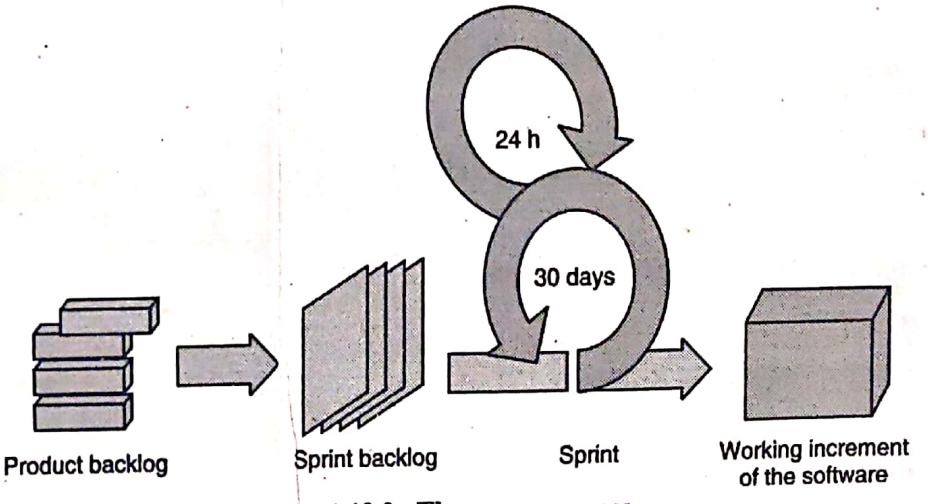


Fig. 1.13.2 : The scrum process

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 :

- **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.
- **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.
- **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.

Syllabus Topic : Kanban Model

1.14 Kanban Model

Q. 1.14.1 Explain the Kanban Model in detail. (Ref. sec. 1.14) **(10 Marks)**

- Kanban is a popular framework used to implement agile software development. It requires real-time communication of capacity and full transparency of work.
- Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time.
- Kanban is enormously prominent among today's agile software teams, but the kanban methodology of work dates back more than 50 years.
- In the late 1940s Toyota began optimizing its engineering processes based on the same model that supermarkets were using to stock their shelves.
- Supermarkets stock just enough products to meet consumer demand, a practice that optimizes the flow between the supermarket and the consumer.
- Because inventory levels match consumption patterns, the supermarket gains significant efficiency in inventory management by decreasing the amount of excess stock it must hold at any given time. Meanwhile, the supermarket can still ensure that the given product a consumer needs is always in stock.

- When Toyota applied this same system to its factory floors, the goal was to better align their massive inventory levels with the actual consumption of materials.
- To communicate capacity levels in real-time on the factory floor (and to suppliers), workers would pass a card, or "kanban", between teams.
- When a bin of materials being used on the production line was emptied, a kanban was passed to the warehouse describing what material was needed, the exact amount of this material, and so on.
- The warehouse would have a new bin of this material waiting, which they would then send to the factory floor, and in turn send their own kanban to the supplier.
- The supplier would also have a bin of this particular material waiting, which it would ship to the warehouse. While the signaling technology of this process has evolved since the 1940s, this same "just in time" (or JIT) manufacturing process is still at the heart of it.

1.14.1 Kanban for Software Teams

- Agile software development teams today are able to leverage these same JIT principles by matching the amount of work in progress (WIP) to the team's capacity.
- This gives teams more flexible planning options, faster output, clearer focus, and transparency throughout the development cycle.
- While the core principles of the framework are timeless and applicable to almost any industry, software development teams have found particular success with the agile practice.
- In part, this is because software teams can begin practicing with little to no overhead once they understand the basic principles.
- Unlike implementing kanban on a factory floor, which would involve changes to physical processes and the addition of substantial materials, the only physical things a software teams need are a board and cards, and even those can be virtual.

1.14.2 Kanban Boards

- The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team.
- While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations.
- Regardless of whether a team's board is physical or digital, their function is to ensure the team's work is visualized, their workflow is standardized, and all blockers and dependencies are immediately identified and resolved.
- A basic kanban board has a three-step workflow: To Do, In Progress, and Done. However, depending on a team's size, structure, and objectives, the workflow can be mapped to meet the unique process of any particular team.

- The kanban methodology relies upon full transparency of work and real-time communication of capacity, therefore the kanban board should be seen as the single source of truth for the team's work.

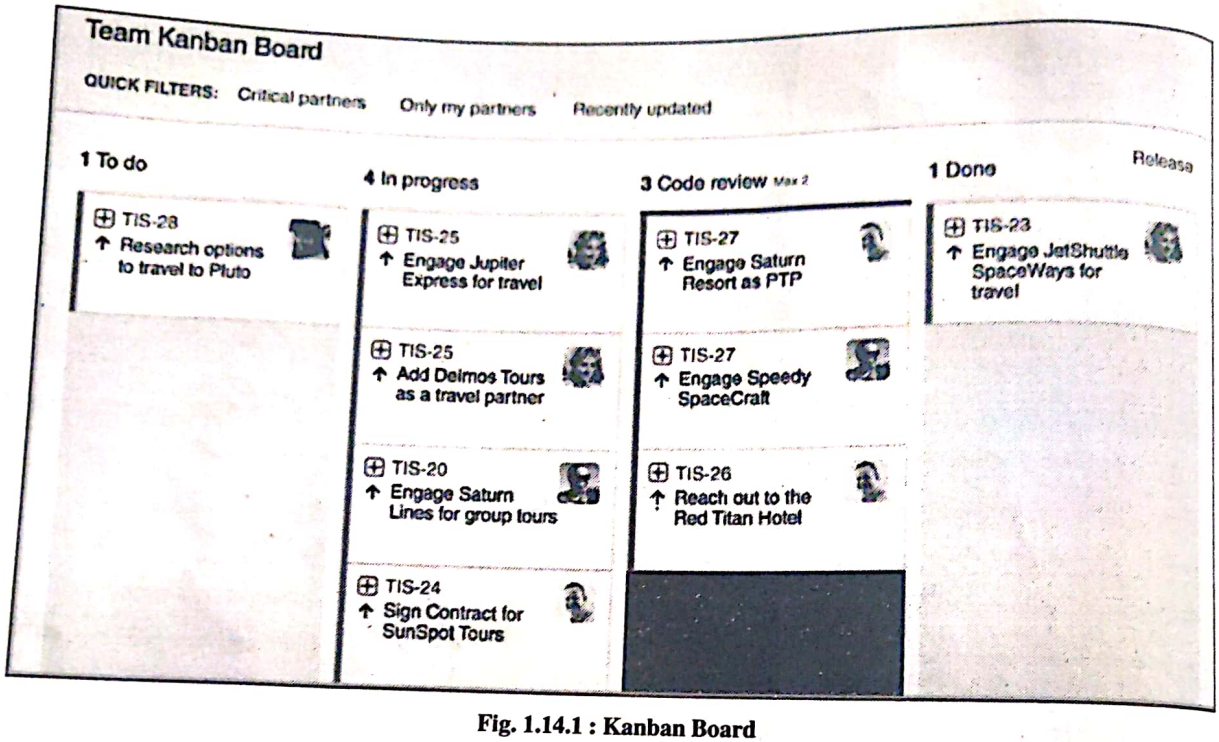


Fig. 1.14.1 : Kanban Board

1.14.3 Kanban Cards

- In Japanese, kanban literally translates to "visual signal." For kanban teams, every work item is represented as a separate card on the board.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner.
- Kanban cards feature critical information about that particular work item, giving the entire team full visibility into who is responsible for that item of work, a brief description of the job being done, how long that piece of work is estimated to take, and so on.
- Cards on virtual kanban boards will often also feature screenshots and other technical details that are valuable to the assignee.
- Allowing team members to see the state of every work item at any given point in time, as well as all of the associated details, ensures increased focus, full traceability, and fast identification of blockers and dependencies.

1.14.4 Advantages of Kanban

- Kanban is one of the most popular software development methodologies adopted by agile teams today.

- Kanban offers several additional advantages to task planning and throughput for teams of all sizes :

Advantages of Kanban

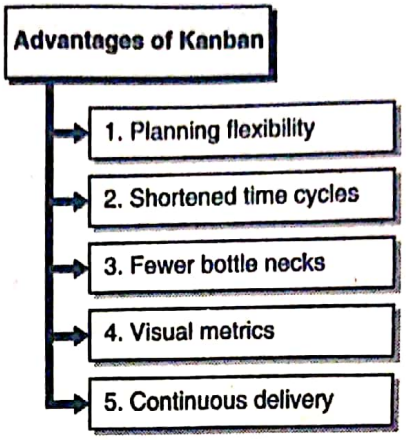


Fig. 1.14.2 : Advantages of Kanban

1. Planning flexibility

- A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog.
- The product owner is free to reprioritize work in the backlog without disrupting the team, because any changes outside the current work items don't impact the team.
- As long as the product owner keeps the most important work items on top of the backlog, the development team is assured that they are delivering maximum value back to the business. So there's no need for the fixed-length iterations that are found in scrum.

2. Shortened time cycles

- Cycle time is a key metric for Kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow - from the moment work starts to the moment it ships.
- By optimizing cycle time, the team can confidently forecast the delivery of future work.
- Overlapping skill sets lead to smaller cycle times. When only one person holds a skill set, that person becomes a bottleneck in the workflow. So teams employ basic best practices like code review and mentoring help to spread knowledge.
- Shared skills mean that team members can take on heterogeneous work, which further optimizes cycle time. It also means that if there is a backup of work, the entire team can swarm on it to get the process flowing smoothly again. For instance, testing isn't only done by QA engineers. Developers pitch in, too.
- In a Kanban framework, it's the entire team's responsibility to ensure work is moving smoothly through the process.

3. Fewer bottlenecks

- Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion.
- That's why a key tenant of Kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.
- For example, a typical software team might have four workflow states: To Do, In Progress, Code Review, and Done.
- They could choose to set a WIP limit of 2 for the code review state. That might seem like a low limit, but there's good reason for it: developers often prefer to write new code, rather than spend time reviewing someone else's work.
- A low limit encourages the team to pay special attention to issues in the review state, and to review others work before raising their own code reviews. This ultimately reduces the overall cycle time.

4. Visual metrics

- One of the core values is a strong focus on continually improving team efficiency and effectiveness with every iteration of work.
- Charts provide a visual mechanism for teams to ensure they're continuing to improve. When the team can see data, it's easier to spot bottlenecks in the process (and remove them).
- Two common reports Kanban teams use are control charts and cumulative flow diagrams.
- A control chart shows the cycle time for each issue as well as a rolling average for the team.

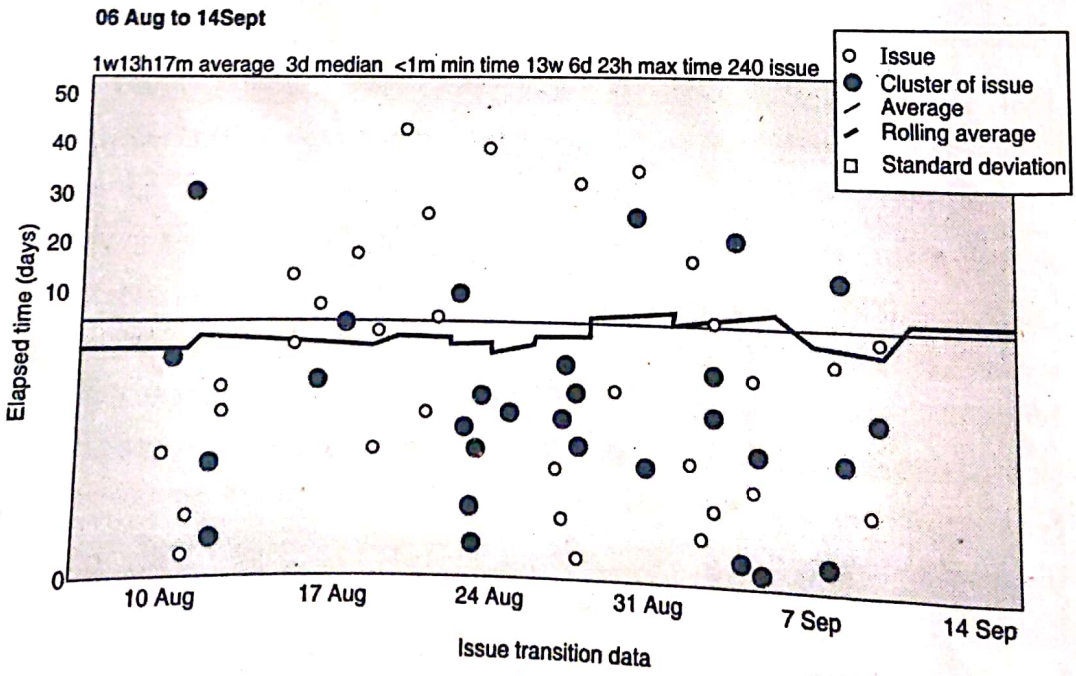


Fig. 1.14.3 : Control Chart

- A cumulative flow diagram shows the number of issues in each state. The team can easily spot blockages by seeing the number of issues increase in any given state. Issues in intermediate states such as "In Progress" or "In Review" are not yet shipped to customers, and a blockage in these states can increase the likelihood of massive integration conflicts when the work does get merged upstream.

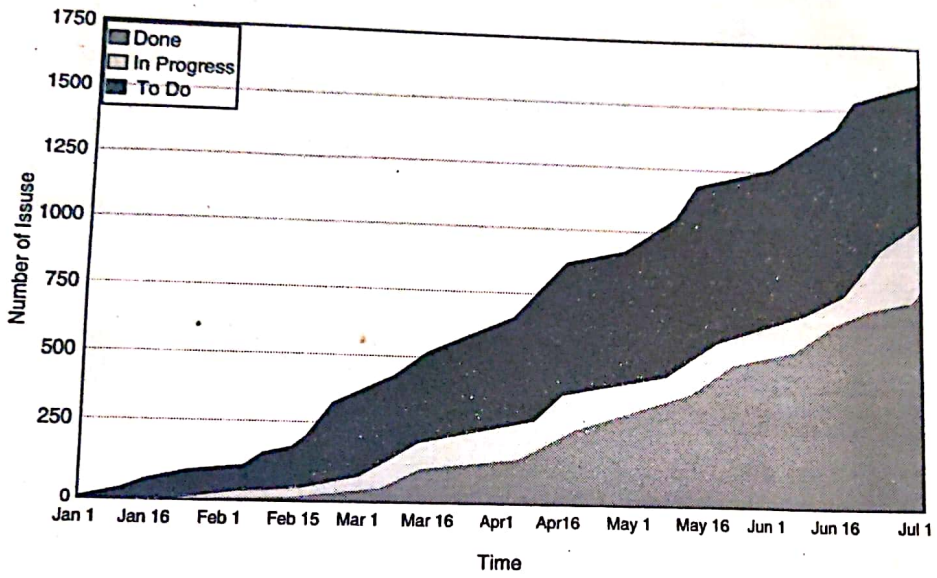


Fig. 1.14.4 : Cumulative Flow Diagram

5. Continuous Delivery

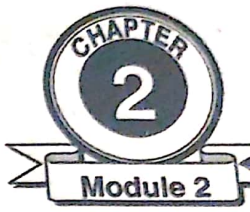
- We know that continuous integration the practice of automatically building and testing code incrementally throughout the day is essential for maintaining quality. Now it's time to meet continuous delivery (CD).
- CD is the practice of releasing work to customers frequently even daily or hourly.
- Kanban and CD beautifully complement each other because both techniques focus on the just-in-time (and one-at-a-time) delivery of value.
- The faster a team can deliver innovation to market, the more competitive their product will be in the marketplace. And Kanban teams focus on exactly that: optimizing the flow of work out to customers.

1.14.5 Comparison of Scrum and Kanban

Q. 1.14.2 Differentiate between Scrum and Kanban Model. (Ref. sec. 1.14.5) (5 Marks)

Kanban and scrum share some of the common concepts but have very different approaches. They should not be confused with one another.

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.



Syllabus

Requirement Elicitat

Requirement Model

NBAN

ry or at the team's

ome teams enlist the h.

at any time.

Chapter Ends...

❑❑❑

CHAPTER 2

Module 2

Requirements Analysis and Modelling

Syllabus

Requirement Elicitation, Software requirement specification (SRS), Developing Use Cases (UML).

Requirement Model : Scenario-based model, Class-based model, Behavioural model.

2.1 Requirement Engineering

→ (MU - Dec. 16)

Q. 2.1.1 What do you mean by requirement? (Ref. sec. 2.1) **Dec. 16, 2 Marks**

☞ **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 :

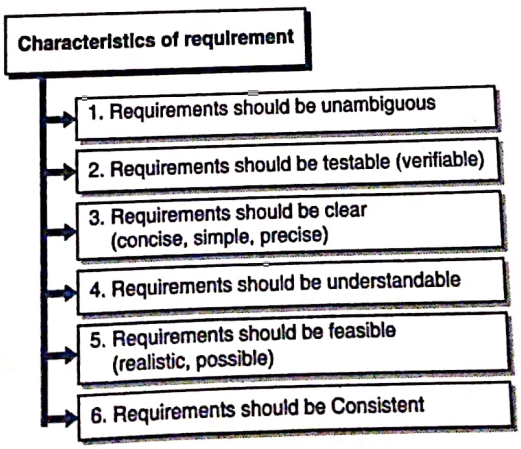


Fig. 2.1.1 : Characteristics of requirements

1. Requirements should be unambiguous

- Ambiguous means the single word or statement has more than one meaning. If requirem 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 require whether they are implemented successfully or not. For easy test, the requirement should be 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 appro fulfillment of the requirement.

4. Requirements should be understandable

- The requirements should be understandable means when anyone read it then it should be 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, estimated budget and time.

6. Requirements should be Consistent

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

Requirement Engineering

- The procedure which collects the software requirements from customer, analyze and docum them is called as **requirement engineering.**
- The aim of requirement engineering is to create and maintain 'System Requirem Specification' document.
- Requirements engineering is the process of understanding and defining which services required and identifying the constraints on these services.
- Requirement engineering processes ensures your software will meet the user expectations ending up with high quality software.

- It's a critical stage of the software process as errors at this stage will reflect later on in the next stages, which definitely will cause you higher costs.
- At the end of this stage, a requirement document that specifies the requirements will be produced and validated with the stakeholders.

2.1.1 Activities Involved In Requirements Engineering

Q. 2.1.2 What are major tasks of requirement engineering? (Ref. sec. 2.1.1) (5 Marks)

- The **activities/tasks** involved **in requirements engineering** vary widely, depending on the type of system being developed and the specific practices of the organization(s) involved.
- These may include :

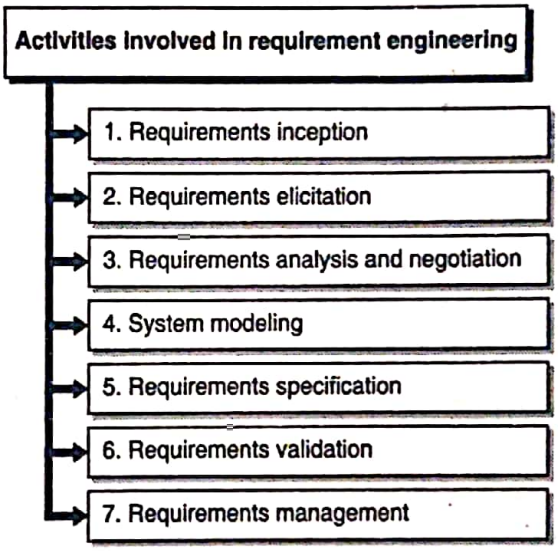


Fig. 2.1.2 : Activities involved in requirements engineering

➜ **1. Requirements Inception**

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

➜ **2. Requirements Elicitation**

- 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".
- We will see more details regarding requirement gathering in section 2.2.

➜ **3. Requirements Analysis and Negotiation**

- Requirements are identified (including new ones if the development is iterative) and conflicts with stakeholders are solved. Both written and graphical tools (the latter commonly used in the design phase but some find them helpful at this stage, too) are successfully used as aids.
- Examples of written analysis tools : use cases and user stories.
- Examples of graphical tools : UML and LML.

➜ **4. System modelling**

- Some engineering fields (or specific situations) require the product to be completely designed and modeled before its construction or fabrication starts and, therefore, the design phase must be performed in advance.
- For instance, blueprints for a building must be elaborated before any contract can be approved and signed.
- Many fields might derive models of the system with the Lifecycle Modeling Language, whereas others might use UML.

Note : In many fields, such as software engineering, most modeling activities are classified as design activities and not as requirement engineering activities.

➜ **5. Requirements specification**

- Requirements are documented in a formal artifact called Requirements Specification (RS).
- Nevertheless, it will become official only after validation. A RS can contain both written and graphical (models) information if necessary.
- Example : Software Requirements Specification (SRS).

➜ **6. Requirements validation**

It is the process of checking whether the documented requirements and models are consistent and meet the needs of the stakeholder. Only if the final draft passes the validation process, the RS becomes official.

➜ **7. Requirements management**

Managing all the activities related to the requirements since inception, supervising as the system is developed and, even until after it is put into use (e. g., changes, extensions, etc.)

Syllabus Topic : Requirement Elicitation

2.2 Requirement Elicitation

➜ (MU - Dec. 17

Q. 2.2.1 Mention any four requirement Elicitation Methods. (Ref. sec. 2.2) Dec. 17, 5 Marks

- 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 :
  - Structured or closed interviews in which information to be collected from customer is decided in advance.
  - Non-structured or open interviews in which details needs to be collected from customer are decided in advance.
  - Oral interviews.
  - Written interviews.
  - Face-to-face interviews which are held among two people across the table.
  - 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 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 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 detailed 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.

2.3 Requirement Analysis

Q. 2.3.1 Write note on Requirements analysis. (Ref. sec. 2.3) (5 Marks)

- Requirements analysis is vital phase of requirement engineering process after Elicitation.
- In this activity, we understand and refine the collected requirements to make them consistent and non-confusing.
- This activity reviews each and every requirement and may also give graphical view of the overall system.
- After the analysis, it is supposed that the understanding of requirments of project are increased significantly.
- In this activity, developer can communicate with customer to clear confusing points and to understand which requirments are more vital than others.
- The significant factors in the requirement analysis activity are :
  1. Identify and solve confusion among requirements at the same level and among different levels.
  2. Identify the boundaries of the proposed software system and the way in which it communicate with environment.
  3. Evaluate customer requirements for overall system requirements and then break down them into individual component level.
- The objective of the Requirement Analysis is generating a solution to implement the requirements.
- Process of the Requirements Analysis includes following steps :
  1. Analysis of the requirements
  2. Description of the solution
  3. Cost estimate and prioritization.
- Here are some examples of how we can consider problems occurred in requirement analysis.
- Stakeholders don't recognize what they really required.
- Stakeholders state the requirements in their own words.
- Various stakeholders may have different and confusing requirements.
- Organizational and political factors may affect the software needs.
- The requirements are modified at the time of requirement analysis process.
- New stockholders may be included and the business environment may change.

2.4 Types of Requirements

Q. 2.4.1 Explain the types of requirements. (Ref. sec. 2.4) (10 Marks)

- The requirements, which are in general assumed, are divided into three categories, functional requirements, non-functional requirements, and domain requirements.

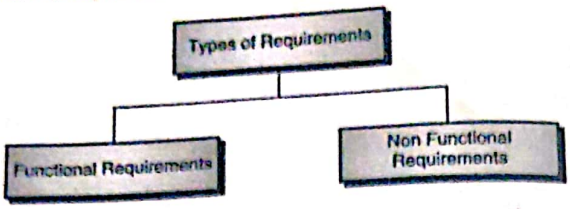


Fig. 2.4.1 : Types of requirements

2.4.1 Functional Requirements

Q. 2.4.2 Explain Functional Requirements. (Ref. sec. 2.4.1) Dec. 19, (MU...

- The Functional requirements specification create document regarding the operations and 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, 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 the 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 :**
  - Descriptions of data to be entered into the system.
  - Descriptions of operations performed by each screen.
  - Descriptions of work-flows performed by the system.
  - Descriptions of system reports or other outputs.
  - Persons who can enter the data into the system.
  - Way the system meets applicable regulatory requirements.

☛ **Examples of Functional Requirements**

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

☛ **Interface requirements**

- Field 1 accepts numeric data entry.
- Field 2 only accepts dates before the cur
- Screen 1 can print on-screen data to the

☛ **Business Requirements**

- Data must be entered before a request
- Clicking the Approve button moves the
- All personnel using the system will be

☛ **Regulatory/Compliance Requirements**

- The database will have a functional au
- The system will limit access to author
- The spreadsheet can secure data with

☛ **Security Requirements**

- Members of the Data Entry group can
- Members of the Managers group can
- Members of the Administrators group

2.4.2 Non Functional Requirements (NFR

Q. 2.4.3 Explain Non-Functional Requirem

- These are basically the quality con contract.
- The priority or extent to which th They are also called non-behavioral
  - Portability
  - Security
  - Maintainability
  - Reliability
  - Scalability
  - Performance
  - Reusability
  - Flexibility
- NFRs are classified into followin
  - Interface constraints
  - Performance constraints :

- **Interface requirements**
  - Field 1 accepts numeric data entry.
  - Field 2 only accepts dates before the current date.
  - Screen 1 can print on-screen data to the printer.
- **Business Requirements**
  - Data must be entered before a request can be approved.
  - Clicking the Approve button moves the request to the Approval Workflow.
  - All personnel using the system will be trained.
- **Regulatory/Compliance Requirements**
  - The database will have a functional audit trail.
  - The system will limit access to authorized users.
  - The spreadsheet can secure data with electronic signatures.
- **Security Requirements**
  - Members of the Data Entry group can enter requests but cannot approve or delete requests.
  - Members of the Managers group can enter or approve a request but cannot delete requests.
  - Members of the Administrators group cannot enter or approve requests but can delete requests.

2.4.2 Non Functional Requirements (NFR)

→ (MU - Dec. 16)

Q. 2.4.3 Explain Non-Functional Requirements with types. (Ref. sec. 2.4.2) Dec. 16. 4 Marks

- 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:
  - Portability
  - Security
  - Maintainability
  - Reliability
  - Scalability
  - Performance
  - Reusability
  - Flexibility
- NFRs are classified into following types :
  - Interface constraints
  - Performance constraints : response time, security, storage space, etc.

- Operating constraints
- Life cycle constraints : maintainability, portability, etc.
- 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.

2.4.2(A) Types of Non-Functional Requirements

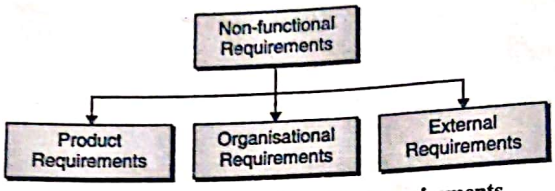


Fig. 2.4.2 : Types of non-functional requirements

– **Product requirements :** Requirements which mention that the delivered product should have behavior in a specific way, e.g. execution speed, reliability etc.

– **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.

– **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.

2.4.3 Domain Requirements

– Domain requirements are used to describe system characteristics as well as features which impact the domain. Those may be new functional requirements, constraints on present requirements or may define particular computations.

– If domain requirements are not satisfied, the system may be unworkable.

– Example : Library system

Syllabus Topic : Software Requirement Specification (SRS)

2.5 Software Requirement Specification (SRS)

→ (MU - May 15

Q. 2.5.1 What is SRS document ? (Ref. sec. 2.5) **May 15, 4 Marks**

– 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.

2.5.1 Advantages of SRS

Q. 2.5.2 What are advantages of SRS? (Ref. sec. 2.5.1) **(4 Marks)**

(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.

2.5.2 Characteristics of SRS

Q. 2.5.3 Explain any five characteristics of SRS. (Ref. sec. 2.5.2) **(10 Marks)**

A good software requirement specification must satisfy following characteristics.

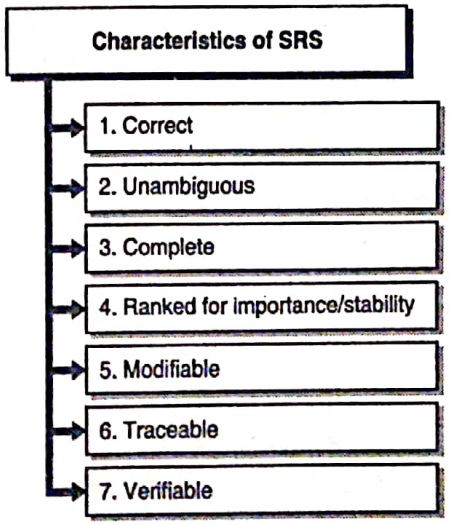


Fig. 2.5.1 : Characteristics of SRS

➜ **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.
- Remember that clear requirement is needed for verifiability.

2.5.3 Format of SRS

Q. 2.5.4 Explain general format of Software Requirement Specification (SRS). (Ref. sec. 2.5.3) **(10 Marks)**

Structure of SRS document includes following sections with various subsections in it :

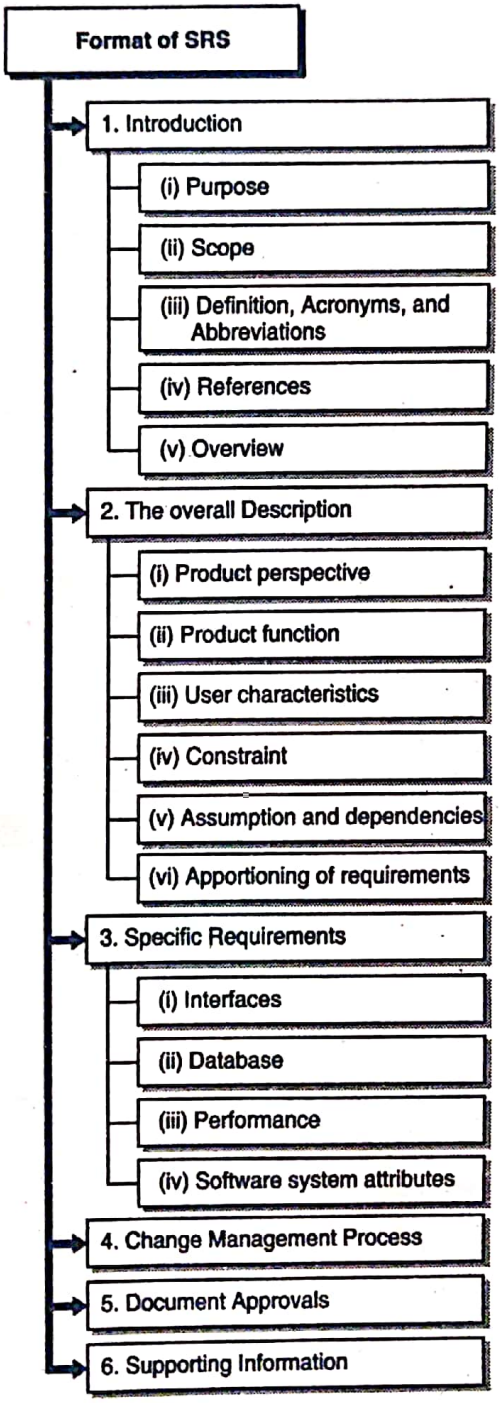


Fig. 2.5.2 : Format of SRS

1. Introduction

Introduction contains brief information about the SRS. This part is further divided into following subsections :

(i) Purpose

This section specifies the main purpose of SRS document with its intended audience for which SRS is constructed.

(ii) Scope

- This section specifies scope of the system for which SRS is built.
- Scope defines not only benefits, objectives and behaviour of software product but also limitations of software so as to understand the boundary of software.

(iii) Definition, Acronyms, and Abbreviations

To avoid ambiguity of terms specified in requirements, SRS provides definition, abbreviation and acronyms about these terms.

(iv) References

It contains list of all documents which are referred by this document.

(v) Overview

It gives overview of the document including its goals and objectives of the system.

2. The overall Description

It provides overall information about the requirements. Customers/users of the system are concerned about this section. It contains following subsections :

(i) Product perspective

This section contains the information which states the benefits of current product over other existing products.

(ii) Product function

- The functionality of software product summarizes in this section of SRS document.
- It is possible to use diagrams to summarize the software functionality and logical relationship among variables present in the system. This information is provided in simple way so that users of the system can understand.

(iii) User characteristics

To use any system, it is mandatory that users of system should have some basic knowledge i.e. at least the educational qualification or basic knowledge related to the field. This section provides the criteria about the users of the system.

(iv) Constraint

It includes limitations of components used in the system for example, hardware limitations.

(v) Assumption and dependencies

It contains the list of factors on which SRS is depending. That is if the factor changes it leads to change in the SRS too.

(vi) Apportioning of requirements

It states the order in which the specified requirements are fulfilled.

➜ **3. Specific Requirements**

This section of SRS actually specifies the Requirements briefly by taking help of information stated in above section. This section is divided into following subsections :

(i) Interfaces

- This section of SRS contain the information about various interfaces that are needed for product, like hardware/software interface, system interface, user interfaces, and communication interfaces.
- It means that each and every input device, output device and the communication medium used by system is introduced in this section of SRS document.

(ii) Database

To store data required and generated by system, database is required. Information about the database used in the system is given in this section.

(iii) Performance

It describes the system performance which includes the required speed, task completion time, response time and throughput of the system.

(iv) Software system attributes

- System attributes are : reliability, security, maintainability.
- This section of SRS provides the assurance that all system attributes will be provided by product and the way in which terms these attributes are satisfied/ fulfilled.

➜ **4. Change Management Process**

- Due to additional user requirements, changes in the system are occurred. This section provides a way that will help to handle such kind of changes.
- As the system make any changes according to user requirements, SRS document should be changed accordingly.

➜ **5. Document Approvals**

- The SRS document should be accepted by both the parties, including the customers for whom the system is developing and the developer who develop the system.

- Document Approvals section include the approval date time and sign of both the parties.

→ **6. Supporting Information**

- It includes guidelines about how to use SRS, index of SRS, references, etc.

2.5.4 SRS Document for Online Student Feedback System

→ (MU - May

Q. 2.5.5 Build an SRS document for online student feedback system. (Ref. sec. 2.5.4) May 15, 5 Mark

Table of Contents

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Overview
2. General Description
   - 2.1 User Manual
3. Functional Requirements
   - 3.1 Description
4. Interface Requirements
   - 4.1 GUI
   - 4.2 Hardware Interface
   - 4.3 Software Interface
5. Performance Requirements
6. Design Constraints
7. Other non-functional Attributes
   - 7.1 Security
   - 7.2 Reliability
   - 7.3 Availability
   - 7.4 Maintainability
   - 7.5 Reusability
8. Operational Scenarios
9. Preliminary Schedule

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.

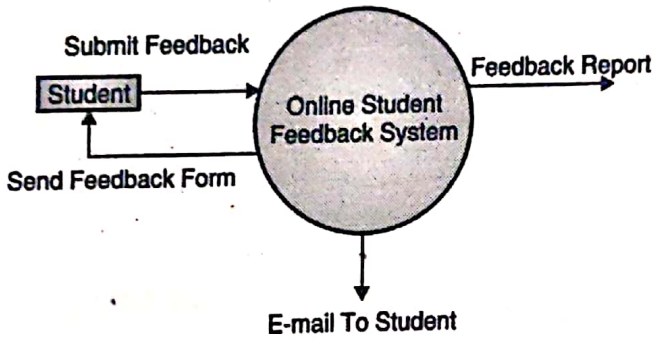


Fig. 2.5.3 : Online student feedback system

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.

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.

2.5.5 SRS for Hospital Management System

→ (MU - May 17, May 18)

Q. 2.5.6 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 (Ref. sec. 2.5.5)

May 17, May 18, 20 Marks

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 :

- Password Management
- Regular Database Archiving
- 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 :

- Accurate and hence reliable.
- Secured.
- Fast Speed.
- Compatibility.

Syllabus Topic : Developing Use Cases (UML)

2.6 Developing Use Cases (UML)

Q. 2.6.1 Explain the development of use cases. (Ref. sec. 2.6) **(5 Marks)**

- Use case is a term used in system analysis to determine, clarify and integrate all system requirements.
- It describes how user interacts with the system to achieve certain goal.
- Use case consists of three basic elements as actor, system and goal.

- A use case diagram describes various business activities carried out by a system.

☞ **Components of use case diagram**

- Use case diagram consists of following three main components.

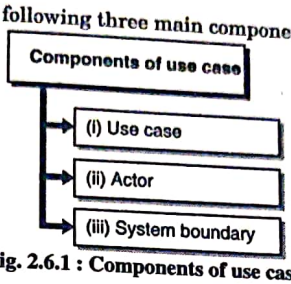


Fig. 2.6.1 : Components of use case

➜ **(i) Use case**

- Use case of a use case diagram represents various business activities performed in a system.
- All discrete business activities of a system can be modeled using use case.
- To identify use cases of a system one should list all discrete business activities of system in problem statement.
- It is represented by an elliptical shape labeled with use_case name.
- Example,

Fig. 2.6.2 : Use case

➜ **(ii) Actor**

- An actor is any entity or real world object which performs different functions in the given system.
- An actor in use case diagram interacts with use case of use case diagram.
- To identify actors of system one should search in problem statement of system for the term describing various roles in system.
- An actor is represented by stick figure outside the system boundary.

Fig. 2.6.3 : Actor

➜ **(iii) System boundary**

- System boundary defines the scope of system or limits of system.
- It is representation of entire system as described in problem statement.
- System boundary is represented by solid line rectangular box.

- Use cases are drawn within system boundary where as actor is outside of the system boundary.

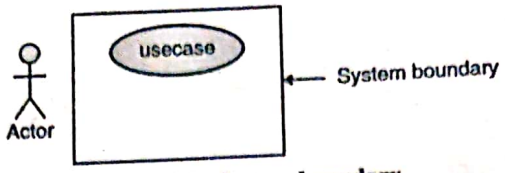


Fig. 2.6.4 : System boundary

☞ **Relationships in use case diagram**

- A relationship between two use cases is in general a dependency among them.
- Use case relationships can be one of the following :

(i) Include

- When a use case is represented as making use of functionality of another use case diagram, the relationship between use cases is called **include** relationship.
- An include relationship is denoted by dotted arrow with arrow head pointing towards derived use case.
- Stereotype <<include>> is labeled on arrow.

Example,

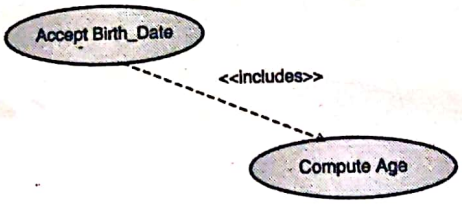


Fig. 2.6.5

(ii) Extend

- The relationship between two use cases in which child use case adds new features to existing functionality of parent use case is called extend relationship.
- It is represented by dotted arrow with arrow head pointing towards parent use case.
- Stereotype <<extend>> is labeled on arrow.

Example,

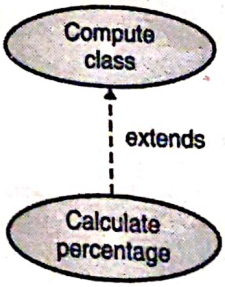


Fig. 2.6.6

- Here percentage helps to decide class of student.

(iii) Generalization

- It is parent-child relationship. Child use case is an underlying process of system but it enhances parent use case.
- It is represented by arrow with triangular arrow head pointing towards parent use case.

Example,

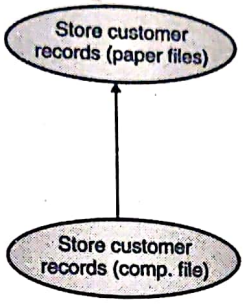


Fig. 2.6.7

The Use Case Model

- A Use Case Model describes the proposed functionality of a new system. A Use Case represents a discrete unit of interaction between a user, called an Actor, (human or machine) and the system.
- This interaction is a single unit of meaningful work, such as Create Account or View Account Details. Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior.

Syllabus Topic : Requirement Model

2.7 Requirement Model

Q. 2.7.1 Explain requirement model. (Ref. sec. 2.7) **(10 Marks)**

- Requirement modeling is implemented after the requirements and constraints have been collected and further analysed.
- Requirement modeling is considered as vital activity to make sure the consistency as well as completeness of the requirements.
- There are different options to model functional, quality attributes and constraints. Selection of suitable approach is depending upon the type of system and the organizational standards.

There are many ways requirements can be modelled; some of the most common requirement models which are as follows :

(a) The Domain Model

- The Domain Model captures all the respective things (Domain Concepts) the product or system "deals with", i.e. represents and manipulates from a business perspective.

- The information captured for a Domain Concept is a description and often properties and their relationship to each other.
- The purpose of the Domain Model is to establish a common understanding of the static aspects of the business domain among all the stakeholders.
- The Domain Model supports the creation of quality requirements by establishing a common vocabulary across all stakeholders reducing ambiguity and redundancy.
- As a result, the following guidelines should be considered :
  - The contents should be written in the business language of the domain and avoid implementation specific or technical aspects (such as operations or actions, which some presentations do not rule out): Use good business names!
  - Stakeholders should be comfortable with the chosen representation of the Domain Model in order to be able to provide feedback. As a result, multiple views on the domain model may be necessary for different stakeholders.
  - The domain model is developed in iterations where necessary details are enriched over time.
- An obvious way to identify Domain Concepts is to identify nouns and phrases in text descriptions of a domain. The information about the Domain Concepts can come from existing documentation, industry standard documents for a specific domain, and output from the requirements elicitation.
- The Domain Model can be represented in different ways. The representation is dependent on methodology and the formality employed in the project as well as the experience of the stakeholders exposed to the Domain Model. It can stretch from a Word document, an Excel table and diagrammatic representations to a fully detailed model representation using a UML class diagram.
- The Domain Driven Design approach is an extension of the Domain Model to produce an implementation which is linked to an evolving model of the domain, it is appropriate for a highly complex domain and produces a highly maintainable system, and the implementation can be done by Domain Specific Language.

(b) The Use Case Model

- A Use Case Model describes the proposed functionality of a new system.
- A Use Case represents a discrete unit of interaction between a user, called an Actor, (human or machine) and the system.
- This interaction is a single unit of meaningful work, such as Create Account or View Account Details.
- Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior.

Some other models used to capture functionality include :

(c) Life Cycles

- These are descriptions of the valid states in each Domain Concept's "life", along with the relationships between those states (e.g. legal transitions).
- It is important to note that such Life Cycles are related to a single Domain Concept (i.e. not to the system as a whole, one or more Use Cases, or the interaction of the system with the outside world).
- Life Cycles are often referred to as State Machines or State Charts.

(d) Business Process models

- The use of Business Process Modelling (BPM) to represent the functionality of the enterprise is an alternative approach for developing software as fundamental concept. It t is to develop the model in a tool which can be then used to produce an executable model.
- Each element has KPIs (Key performance indicators) attached to it so that it can be frequently used as part of a performance improvement process.

(e) User Stories

- In agile projects user stories are used to capture requirements, these are short descriptions of how a particular user wants to interact with a system, in the form "As I want to, so that".
- The user story includes a set of acceptance criteria which describe the boundary of the story and defines when it is done.

(f) Traceability Matrix

- To fulfill the need for traceability a traceability matrix can be produced.
- This can be as simple as a table which cross references the requirements to the software component that fulfils it, however this can become very hard to maintain, so it is more common to use a requirements management tool when this level of formality is required.

2.8 Analysis Modelling

Q. 2.8.1 Write importance of analysis modeling. (Ref. sec. 2.8) **(5 Marks)**

- After the Requirement Analysis, as output gets the specification regarding software's operational characteristics, designates software's interface with other system elements and set up constraints which the specific software must need.
- All the way through analysis modeling the primary focus of software engineer is on *what* and *how*.
- Requirement Analysis helps the software engineer to :
  - Get detailed information of basic requirements which are set in earlier requirement engineering tasks.

- Establish models which describe user scenarios, functional activities, problem classes with their relationships, system and class behavior, and the data flow as it is transformed.

– There are three primary objectives of analysis model :

1. Describe customer needs
2. Establish a basis for the software design generation, and
3. Define a set of requirements which can be validated after creation of software.

– The **analysis model** fills the gap between a **system-level description** which elaborates overall functionality as it is accomplished by the process of applying software, hardware, data, human, and other system elements and a **software design** which elaborates the software application architecture.

– That means Analysis model plays a role of link in between the 'system description' and the 'design model'.

A Bridge

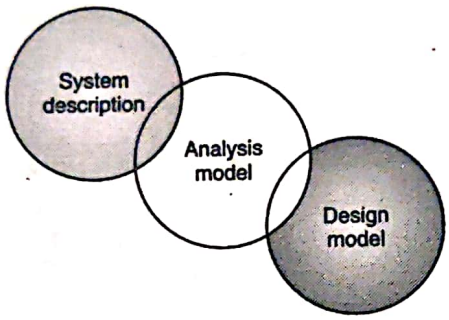


Fig. 2.8.1 : Analysis Model

– In this model, information, functions as well as behaviour of the system is defined and all of the elements of 'design modeling' are translated into the architecture, interface and component level design.

2.8.1 Analysis Rules of Thumb

– The model must concentrate on requirements which are visible within the problem or business domain. The abstraction level must be comparatively high.

– Each and every element regarding the analysis model must be added to an overall understanding of software requirements and supply insight into the information domain, function and behavior of the system.

– Delay consideration of infrastructure and remaining non-functional models until design.

- For example, there may be need of a database, but the classes required to implement it, the functions necessary to access it, and the behavior which will be exhibited after its use must be considered after the completion of problem domain analysis.

– Minimize coupling throughout the system.

- The level of interconnectedness among classes and functions must be minimized.

- Ensure that the analysis model offers value to all the relevant stakeholders.
  - Each constituency has its own use for the model.
- Keep the possible simplest form of the model.
  - Do not introduce extra diagrams if they do not provide any new information.
  - Implement only those modeling elements which have values.

2.8.2 Elements of Analysis Model

Q. 2.8.2 Explain the various elements of analysis modeling in detail. (Ref. sec. 2.8.2) **(5 Marks)**

1. Scenario based element

- Scenario based elements represent the system user point of view.
- **Examples :** Use case diagram, user stories.

2. Class based elements

- The object of class based elements is manipulated by the system.
- It defines the object, attributes and relationship.
- The collaboration is happening among the classes.
- **Examples :** Class diagram, collaboration diagram.

3. Behavioral elements

- These types of elements represent state of the system and how external events affect it.
- **Examples :** Sequence diagram, state diagram.

4. Flow oriented elements

- Information is transferred via a computer-based system.
- It shows way of transforming data objects when they flow between the different system functions.
- **Example :** Data flow diagram, control flow diagram.

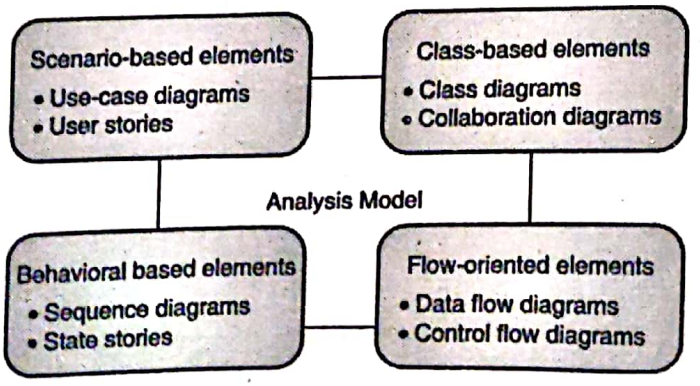


Fig. 2.8.2 : Elements of Analysis Model

Syllabus Topic : Scenario-based Model

2.9 Scenario-based Model

Q. 2.9.1 Write note on Scenario based model. (Ref. sec. 2.9) (10 Marks)

- Use-cases are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)." Ivar Jacobson
- The concept is relatively easy to understand- describe a specific usage scenario in straightforward language from the point of view of a defined actor.

2.9.1 Writing Use-Cases

We have already seen how to develop use cases. The important aspects are :

(1) What should we write about?

Inception and elicitation provide us the information we need to begin writing use cases.

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

Use Cases

- A scenario that describes a "thread of usage" for a system.
- ***Actors*** represent roles played by people or devices as the system functions.
- ***Users*** can play a number of different roles for a given scenario.

2.9.2 Developing an Activity Diagram

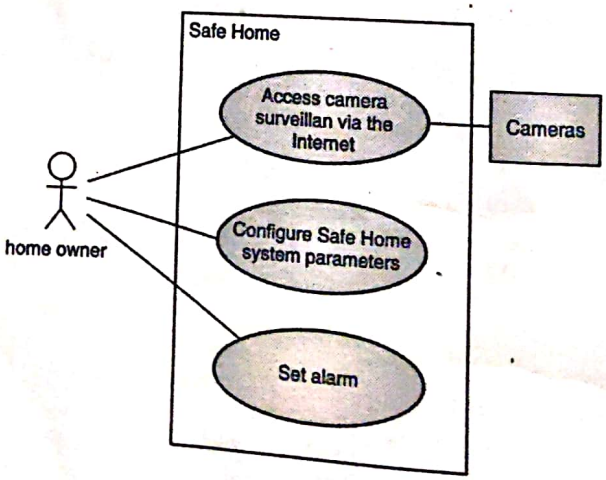


Fig. 2.9.1 : Developing Activity Diagram

- What are the main tasks or functions that are performed by the actor?

- What system information will the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

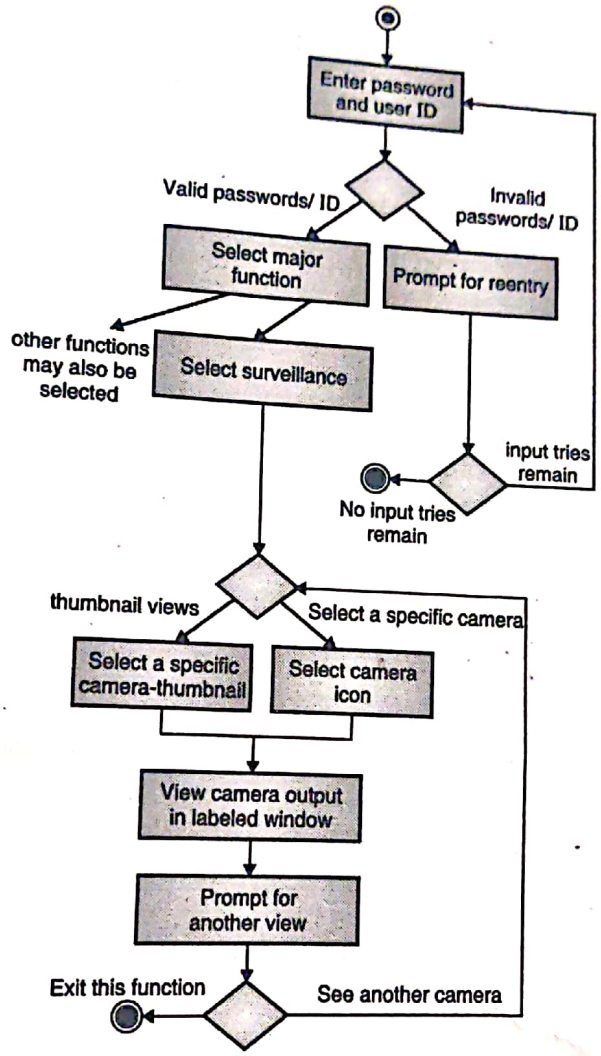


Fig. 2.9.2 : Developing Activity Diagram

2.9.3 Swimlane Diagrams

- The UML *swimlane diagram* is a useful variation of the activity diagram and allows representing the flow of activities described by the use-case and at the same time indicates which actor or analysis class has responsibility for the action described by an activity rectangle.

- Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.

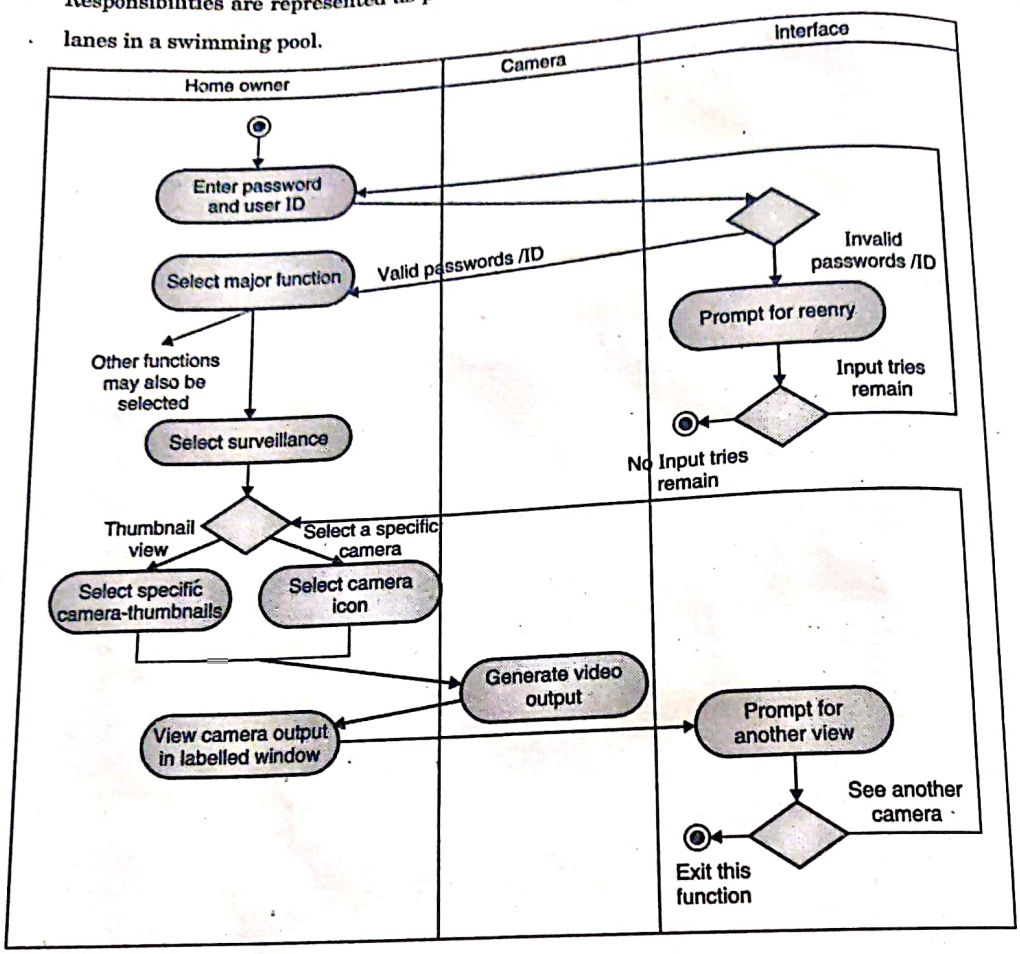


Fig. 2.9.3

Syllabus Topic : Class-based Model

2.10 Class-based Model

Q. 2.10.1 Write note on class based model. (Ref. sec. 2.10) **(10 Marks)**

- This section describes the process of developing an Object-Oriented Analysis (OOA) 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.

2.10.1 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 :
  - **External entities** such as several systems, devices and people that generate or use information to be referred by a computer-based system.
  - **Things** such as reports, displays, letters and signals which are considered as part of the information domain for the respective problem.
  - **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.
  - **Roles** such as manager, engineer and salesperson played by individuals who need to interact with the system.
  - **Organizational units** such as division, group, and team which are basically relevant to an application.
  - **Places** such as manufacturing floor or loading dock that establish the context regarding the respective problem and the overall function of the system.
  - **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 :
  - *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.
  - *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.
  - *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.
  - *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.

- *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.
- *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.

2.10.2 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.

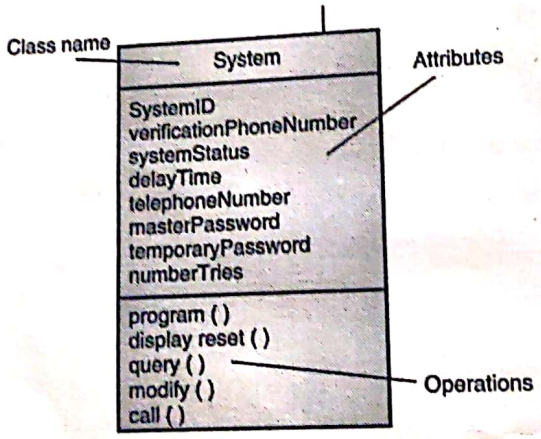


Fig. 2.10.1 : Specifying Attributes

2.10.3 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.

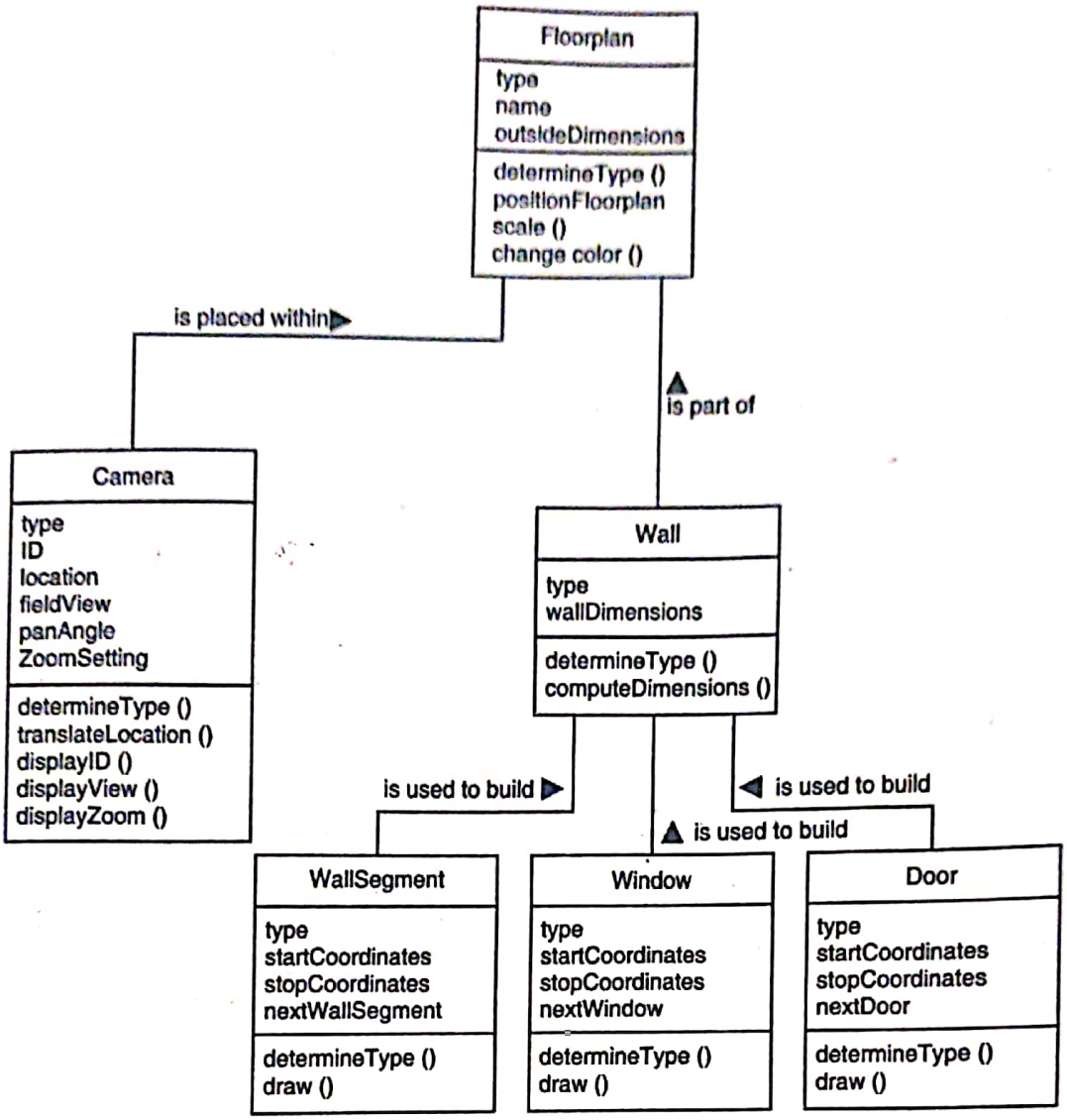


Fig. 2.10.2 : Class Diagram

Syllabus Topic : Behavioural Model

2.11 Behavioural Model

Q. 2.11.1 Explain behavioral based model. (Ref. sec. 2.11) (10 Marks)

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

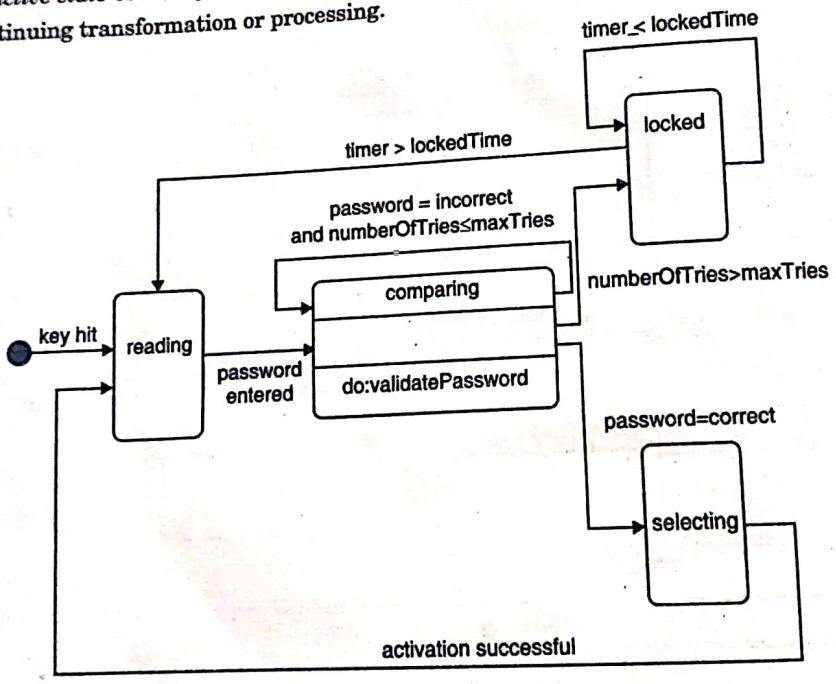
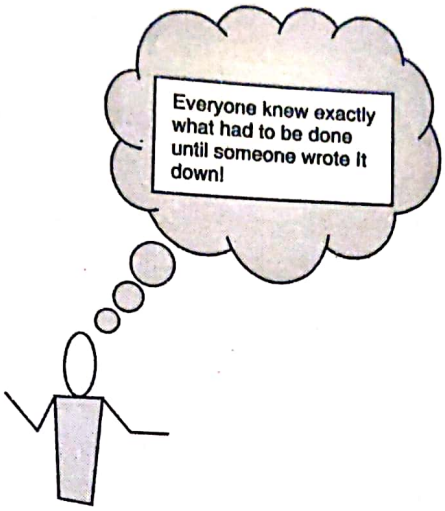


Fig. 2.11.1 : State Representation

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

Writing the Software Specification



☞ **Specification Guidelines**

- Layered format must be used which can provide increasing detail as layer deepen.
- Consistent graphical notation must be used and textual terms should be applied.
- All acronyms should be clearly defined.
- Compulsory include table of content; ideally include an index and glossary.
- Style should be simple and unambiguous.

Chapter Ends...

❑❑❑

CHAPTER 3

Module 3

Project Scheduling and Tracking

Syllabus

Management Spectrum, 3Ps (people, product and process) Process and Project metrics.

Software Project Estimation : LOC, FP, Empirical Estimation Models : COCOMO II Model, Specialized Estimation Techniques.

Project scheduling : Defining a Task Set for the Software Project, Timeline charts, Tracking the Schedule, Earned Value Analysis.

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

Q. 3.1.1 Explain 3 P's in software project spectrum. (Ref. sec. 3.1) **(5 Marks)**

- 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.
- We will see all of those three P's of management spectrum in detail.

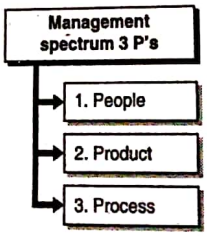


Fig. 3.1.1 : Management spectrum 3 Ps

➜ 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, r
of their software developme
- Enterprises which attain
probability of carrying effi

➜ 2. The Product

- Product is nothing but
development of produc
consideration of subst
constraints must be do
- If there is lack of th
estimates of the cost,
controllable project s

➜ 3. The Process

- A software process
complete plan for s
- There are differen
assurance points
with the function
team.
- At last, umbrell
any one framew

3.2 Process and

- It is possible
on a regular
- Measureme
quality co
- At the end
the quali
project d
- In the
concern

3.2.1 Metrics

- The c
time.

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.

Syllabus Topic : Process and Project Metrics

3.2 Process and Project Metrics

- It is possible to apply measurement to any of the software process for the purpose of improving it on a regular basis.
- Measurement is very useful throughout the lifecycle of a software project to help in estimation, quality control, productivity assessment as well as in project control.
- At the end, the use of measurement is done by the software engineers for the purpose of assessing the quality of respective work products and to help in strategic decision making process as the project development continues.
- In the perspective of projects and the software process conducted by the project, the prime concern of software team is productivity as well as quality metrics.

3.2.1 Metrics in the Process and Project Domain

- The collection of process metrics is done through all the related projects and for a long period of time.

- The basic idea behind is to provide a series of process indicators which results in long-lasting software process improvement.
- Project metrics enable a software project manager to :
  - Assess the status of an ongoing project
  - Track potential risks
  - Uncover problem areas before they go "critical"
  - Adjust work flow or tasks
  - Evaluate the project team's ability to control quality of software work products.

3.2.1(A) Process Metrics and Software Process Improvement

Q. 3.2.1 Write note on Process Metrics. (Ref. sec. 3.2.1(A)) (5 Marks)

- The unique efficient method to make improvement in any process is the measure of particular attributes of the process, establish a set of significant metrics depending upon these attributes and further use the metrics to provide indicators which will result in the strategy for improvement.
- The important factor is that process is considered as only one of several controllable factors in the practice of improving software quality as well as organizational performance.
- Fig. 3.2.1 shows that the process best fits at the center of a triangle in which three factors are connected which have deep impact on software quality as well as organizational performance.
- The skill and motivation of the stakeholders is considered as the single most significant factor in quality and performance.
- There is significant impact of complexity of the product on quality and performance of the team.
- Also one important factor which has significant impact is the technology (methods and tools regarding software engineering).

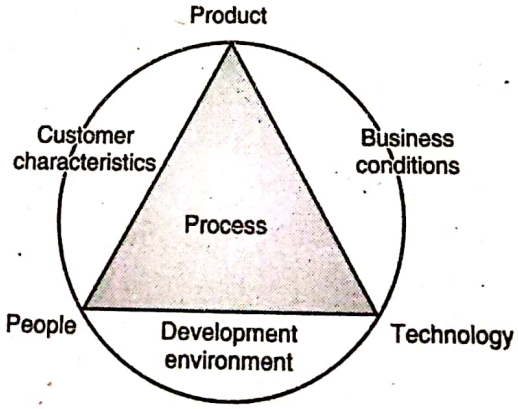


Fig. 3.2.1 : Determinants for software quality and organizational effectiveness

- Additionally, there is a circle of environmental conditions around the process triangle which contains following :
  - Development environment - integrated software tools,

- Business conditions - deadlines, business rules,
- Customer characteristics - ease of communication and collaboration.

- It is possible for user to measure only the effectiveness of a software process in an indirect manner. That means, a set of metrics are derived by the user based on the results which user can derive from the respective process.
- Results or outcomes contains measures of errors which have been found before releasing the software, defects which have been delivered to and further reported by the end users, work products which have been delivered(productivity), human effort spent, duration expended, schedule conformance, and other measures.
- User is also able to derive process metrics by the way of measuring the characteristics regarding particular software engineering tasks.
- For example, user might measure the effort and time which have been exhausted in implementing the umbrella activities as well as the generic software engineering activities.
- Grady (Grady Booch is an American software engineer, best known for developing the UML) argues that there are "private and public" uses for different types of process data.
- Because it is very obvious that individual software engineers might be susceptible in using the metrics gathered on personal basis, these data is expected to be private to the respective individual and work as an indicator for that individual only.
- In the examples of private metrics we can consider defect rates (by individual), defect rates (by component), and bugs found in development process.
- The "private process data" viewpoint is well compatible with the Personal Software Process approach.
- It is recognized that the beginning of software process improvement must be at individual level.
- Private process data is considered as an important aspect in improving the software engineering approach.
- Some process metrics are private to the software project team but might be public to all the remaining team members.
- For examples, errors found regarding main software functionalities, defects reported in technical reviews and LOC (Lines of Code) or FP (Function Points) per component or function.
- These data are reviewed by the team to get indicators to improve team performance.
- Usually public metrics incorporate data which originally was private to individuals as well as for teams.
- Defect rates at project level, effort, duration, and associated information are gathered and evaluated for the purpose of getting indicators which can improve organizational process performance.
- When an enterprise works for the enhancement of its overall level of process maturity, software process metrics plays a significant role.

- However, like all other metrics, there is also possibility of misuse of software process metrics which may leads to generation of more problems than they solve.
- Grady suggests a "software metrics etiquette" which is suitable for both managers as well as practitioners as they organize a process metrics program :
  - Make use of common sense as well as organizational sensitivity in the process of interpreting metrics data.
  - Give usual feedback to the individuals and teams who does the collection of measures and metrics.
  - Don't use metrics for the purpose of appraisal to individuals.
  - Communicate with practitioners and teams to establish clear objectives and metrics which will be helpful to achieve them.
  - Don't refer metrics for the purpose of threatening individuals or teams.
  - Metrics information which specifies a problem area should not be taken as "negative." This information is just an indicator for process improvement.
  - Don't obsess on a single metric to the exclusion of other important metrics.
- As the enterprise becomes more compatible with the gathering and use of process metrics, the origin of simple indicators provides means to a more precise approach known as Statistical Software Process Improvement (SSPI).
- Essentially, SSPI takes reference of software failure analysis for the purpose of gathering data regarding all bugs and defects found as an application, system, or product is developed and used.

3.2.1(B) Project Metrics

Q. 3.2.2 Write note on Project Metrics. (Ref. sec. 3.2.1(B)) (5 Marks)

- Dissimilar to software process metrics which are helpful in strategic purposes, software project measures are considered as tactical.
- It means, project metrics as well as the indicators derived from those project metrics are used by a project manager and the respective software team for the aim of adapting project workflow and technical activities.
- In most of the projects, the first application of project metrics takes place in the process of estimation.
- Metrics which have been gathered from previous projects are used as a basis for new project. Those metrics are useful for estimation of effort and time for current software work.
- As the project proceeds, comparison of measures of effort and time spent is done with original estimates.
- The data is used by the project manager to monitor and control progress.
- As there is beginning of technical work, other project metrics started to have importance.
- The measurement of production rates which have been represented in terms of models generated, review hours, FPs, and delivered source lines is done.

- Additionally, errors found in all of the software engineering tasks are tracked.
- As there is transition of software from requirements to design, gathering of technical metrics is done for the purpose of assessing design quality and to provide indicators which will impact the approach considered to code generation and testing.
- The aim of project metrics is twofold. First, these metrics are helpful in reducing the development schedule by the way of making the adjustments which are important to avoid delays and lessen possible problems as well as risks.
- Second, project metrics are helpful in assessing product quality on regular basis and, when needed, make changes in the technical approach to enhance quality.
- As there is improvement in quality, defects are reduced, and as there is reduction in defect count, the amount of rework which may be necessary in the project also get decreased. This ultimately results in reduction of overall project cost.

Syllabus Topic : Software Project Estimation

3.3 Software Project Estimation

Q. 3.3.1 Explain Software Project Estimation in detail. (Ref. sec. 3.3) **(10 Marks)**

- 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 ?

- To understand this we will take a simple example. 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.

3.3.1 Metrics for Size Estimation

→ (MU - May 15, Dec. 16, May 17)

Q. 3.3.2 Explain size oriented software engineering metrics (Ref. sec. 3.3.1) **May 15, Dec. 16, May 17, 5 Marks**

- 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 3.3.1.
- In Table 3.3.1, 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 3.3.1

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 :
  - Errors per KLOC (Thousands line of code)
  - Defects per KLOC,
  - $ Per KLOC,
  - Pages of documentation per KLOC
- The Size-oriented metrics are not considered as the best solution for measuring the software process.

Syllabus Topic : Line of Code (LOC)

3.3.2 Line of Code (LOC)

Q. 3.3.3 Write note on LOC. (Ref. sec. 3.3.2) **(5 Marks)**

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

Syllabus Topic : Function Points (FP)

3.3.3 Function Points (FP)

→ **(MU - May 17)**

Q. 3.3.4 Explain Function Point Estimation Technique in detail. (Ref. sec. 3.3.3) **May 17, 5 Marks**

- 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 on 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 :
  - Number of user inputs : Each user input should be considered.
  - Number of user outputs : Reports, screens, error messages, etc.
  - Number of user inquiries : An on-line i/p which results on on-line o/p.
  - Number of files : Each logical master file is counted.
  - 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 Simle	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. 3.3.1

- 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.

Syllabus Topic : Empirical Estimation Models

3.4 Empirical Estimation Models

Q. 3.4.1 Explain the Empirical estimation techniques. (Ref. sec. 3.4) **(5 Marks)**

- The Empirical estimation techniques are usually founded on making a studied guess regarding the project parameters.
- If similar products are developed previously, then that experience is always helpful.
- Even though the base of empirical estimation techniques is common sense, several activities involved in estimation have been formalized over the years.
- There are two frequently used empirical estimation techniques namely : Expert judgment technique and Delphi cost estimation.

Expert judgment technique

- In Expert Judgment Technique, an expert analyzes the problem in detail and makes guess regarding the problem size.
- Usually, the expert makes an estimation of the cost regarding the different components (such as modules or subsystems) of the respective system and then integrates them to find out the overall estimate.
- Yet, this technique is exposed to human errors. Additionally, it is likely that the expert may miss few factors accidentally.
- Additionally, an expert who is making the estimate may lack of experience as well as knowledge of all aspects of a project.
- For example, he/she may be familiar with the database and user interface components but may not be conversant regarding the computer communication part.

- Estimation made by group of experts can be considered as more refined form of expert judgment.
- It minimizes factors like individual oversight, lack of familiarity with a specific aspect regarding the project, etc.

☞ **Delphi cost estimation**

- In this approach some of the shortcomings of the expert judgment approach are tried to resolve.
- In this approach a team which involves a group of experts and a coordinator takes part.
- The coordinator gives a copy of the Software Requirements Specification (SRS) document to all of the estimators and a particular form for the purpose of recording their cost estimates.
- Individual estimates are completed by all of the estimators and submitted to the coordinator.
- In individual estimates, the estimators point out any unusual characteristic regarding the product which has great impact on the estimation.
- The coordinator then makes and hand outs the summary of the responses by considering all the estimators, and incorporates any unusual rationale that may be noted by any of the estimators.
- By considering this summary, all of the estimators re-estimate. This method is repeated for several rounds.
- However, estimators cannot communicate with each other regarding the estimates. The reason is that the estimate of more experienced or senior estimator may influence other estimators.
- After several iterations of estimations are done, the coordinator compiles the results and prepares the final estimate.

Syllabus Topic : COCOMO II Model

3.4.1 COCOMO II Model

➔ **(MU - May 15, Dec.16)**

Q. 3.4.2 Explain COCOMO Model. (Ref. sec. 3.4.1) **May 15, Dec. 16, 10 Marks**

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

Table 3.4.1

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. 3.4.1 shows COCOMO II sub-models as well as where they are used.

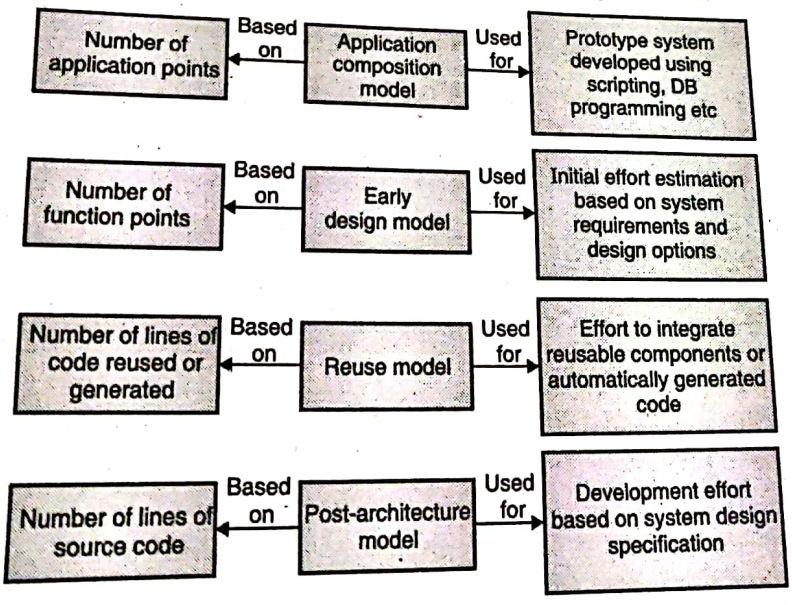


Fig. 3.4.1 : The COCOMO II model

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 3.4.2 shows the levels of object-point productivity.

Table 3.4.2 : 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.
  - **ASLOC :** Number of lines of code in the components that have to be adapted;
  - **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.

Syllabus Topic : Specialized Estimation Techniques

3.5 Specialized Estimation Techniques

Q. 3.5.1 Explain different specialized Estimation Techniques. (Ref. sec. 3.5) (10 Marks)

- 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.
- Here we will see two specialized estimation techniques :

3.5.1 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.

Note : The important elements for estimation can be historical data, an empirical model, or "experience."

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.

3.5.2 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 :
  - Inputs are nothing but the input screen or form (such as CGI or Java), every maintenance screen, each tab (if used).
  - Outputs are every static Web page, every dynamic Web page script (such as ASP, PHP, or other DHTML script), and every report.
  - Tables are every logical table in the DB plus, every XML object (if used).
  - Interfaces preserve their definition as logical files (such as unique record formats) into the out-of-the-system boundaries.
  - Queries are the one which are published externally or take help of a message-oriented interface such as DCOM or COM external references.
- 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.

Syllabus Topic : Project Scheduling

3.6 Project Scheduling

- Project scheduling is a technique that decides the sequence of tasks to be done and assigns organizational resources to complete those tasks in predefined timeframe.
- A project is nothing but a group of many tasks, and each task has start and end dates.
- A project scheduling is a kind of document that summarizes the work required to deliver the project on time.
- It also distributes the estimated efforts across the planned project period.
- In simple words we can say that the project scheduling establishes the "Road Map" for the project manager.

Syllabus Topic : Defining A Task Set for Software Project

3.6.1 Defining A Task Set for Software Project

Q. 3.6.1 Explain the Work Break Down structure in details. (Ref. sec. 3.6.1) (5 Marks)

- The process of dividing complex projects to simpler and manageable work items is called Work Breakdown Structure (WBS).

- Work item is also called as a **Task**. Example of work item is creating a database table with data.
- Project managers use this technique for simplification of project execution. In WBS, larger are broken down to smaller manageable items and later these items are easily verified estimated.
- WBS is not limited to a specific field. This methodology can be used for any type of pro management.

3.6.2 Reasons for creating a WBS In A Project

Q. 3.6.2 Explain the importance of WBS in software engineering with suitable example. (Ref. sec. 3.6.2) (5 Marks)

- Following are the reasons for creating the WBS in project :
  - Do accurate project organization.
  - Assign accurate task of responsibilities to the project team.
  - Do accurate estimation of the cost, time and risk involved in the project.
  - Illustrate the project scope.
  - Plan the project according to availability of resources.

☞ **WBS Construction**

- Starting point of WBS is to identify the main deliverables of a project. Items can be broken do to different levels. One can break down one task to ten items whereas other can break down t same in 20 items.

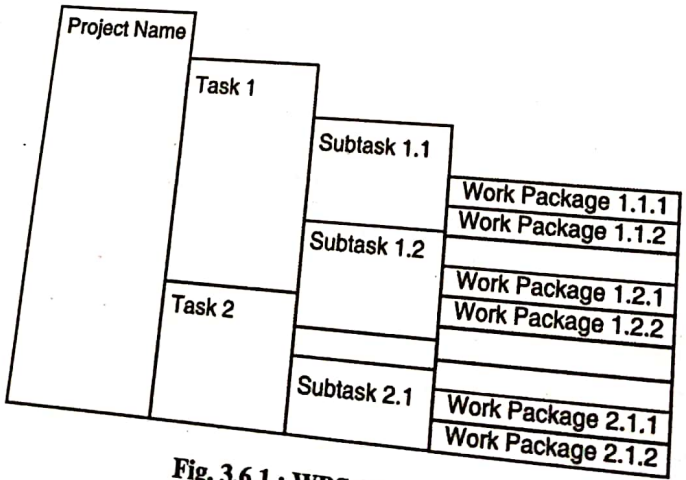


Fig. 3.6.1 : WBS Structure

- The effectiveness of a work breakdown structure can decide the success of a project.
- The WBS provides the base for all project management work like planning, cost, effort estimation, resource allocation, and scheduling.

- The Fig. 3.6.2 shows an example of a WBS for manufacturing of new toy. Here each level is created by breaking down the work items (Decomposition) such as Market Research, Product Design etc.
- Decomposition is the process of breaking down the items into smaller items. The element which is present at the low level is called as Task. In the example below, brochures, advertising and commercials are all work packages or tasks.
- Therefore, creating WBS is critical step in the process of project management.

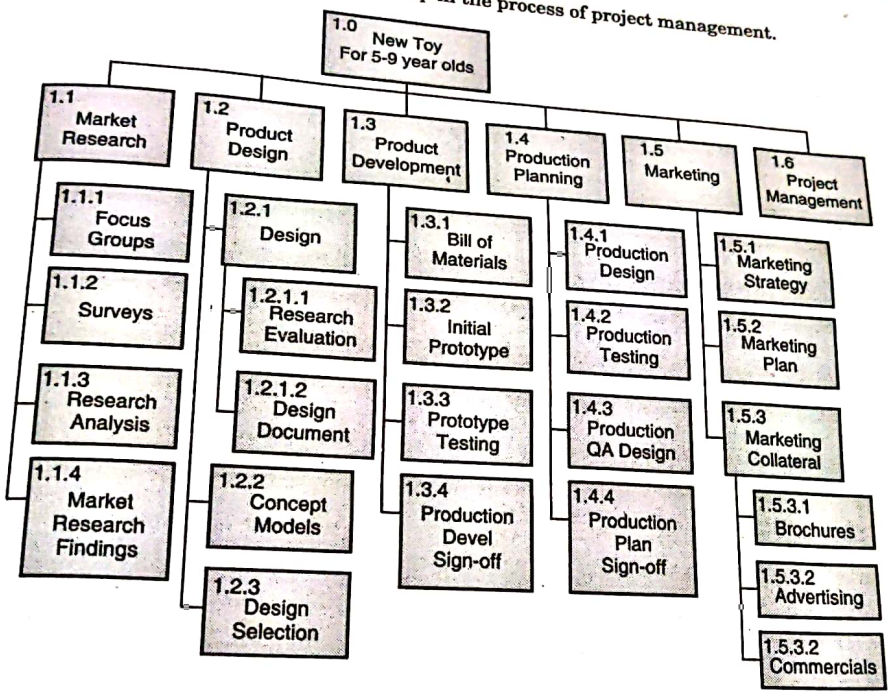


Fig. 3.6.2 : Example of WBS

Benefits of WBS

1. Project budget and schedule can be easily calculated.
2. Helps for identifying the project risk in a given project.
3. If project is falling, the work breakdown structure can find out the major deliverables impacted.

3.7 Scheduling Technique

Q. 3.7.1 Explain the Scheduling in software engineering. (Ref. sec. 3.7) **(5 Marks)**

- Scheduling is generally one of the critical activities for project in any of the field.
- Software project scheduling is nothing but action that distributes estimated effort across the planned project time interval by allocating the effort to specific software engineering tasks.

- It is very important to note that, the schedule evolves over time.
- Macroscopic schedule is developed during preliminary stages of project planning. This type schedule will always identifies all main process activities involved in the project.
- As the project proceeds, each task on the macroscopic schedule is developed into a detailed schedule. Here, specific software actions and tasks are identified and scheduled.
- There are two primary scheduling techniques available in software engineering.
  1. **CPM :** Critical Path Method
  2. **PERT :** Program Evaluation Review Technique.

3.7.1 Critical Path Method

Q. 3.7.2 Write meaning of CPM? Explain with example. (Ref. sec. 3.7.1) (5 Marks)

- 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 complete. 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 activities 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 the 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.

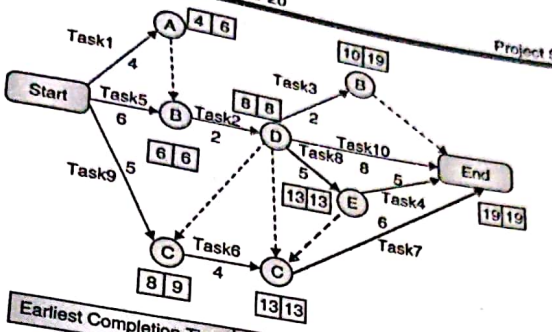


Fig. 3.7.1 : Critical Path

- 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 :
     - Identify the task that takes place before the critical task happens.
     - Identify the tasks that should be completed at the same time.
     - 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 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, 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.

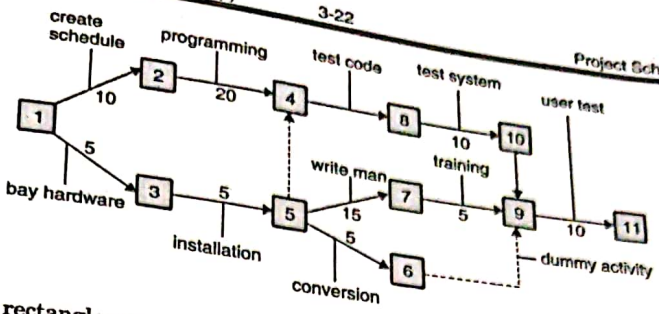
Advantages of Critical Path Method

- Visual representation of the project activities
- To calculate Project deadlines.
- To track the critical activities.

3.7.2 Program Evaluation and Review Technique (PERT)

Q. 3.7.3 Explain PERT. (Ref. sec. 3.7.2) (5 Marks)

- 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. 3.7.2, 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. 3.7.2 : PERT

The Possibilities of the PERT are listed below :

- There are three estimates involved in PERT;

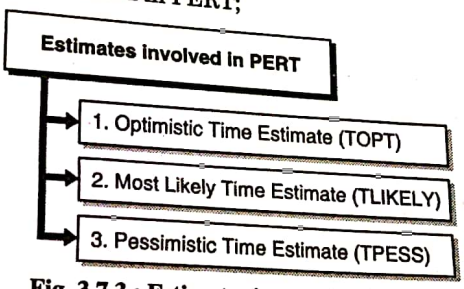


Fig. 3.7.3 : Estimates involved in PERT

- In this way, a range of time is given for each activity with the most probable value, TLIKELY.

Following are further details on each estimate :

➜ 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.

Syllabus Topic : Tracking the Schedule

3.8 Tracking the Schedule

Q. 3.8.1 Explain the term "Tracking Schedule". (Ref. sec. 3.8) (5 Marks)

- The project schedule is a road map which defines the tasks and milestones that are to be tracked and controlled as the project proceeds.
- Tracking can be done in a number of different ways :
  1. By calling project status meetings periodically in which each team member reports progress of assignments/tasks.
  2. Evaluating the reviews during the software engineering process.
  3. Setting the tentative project deadlines that have been completed by the scheduled dates.
  4. Comparing the real start date to the intended start date for every project.
  5. Meeting can be taken informally with resources to obtain their progress on the given assessment.
- All of these tracking techniques are used by experienced project managers.
- If things are going well (i.e., the project is on schedule and within allocated budget, reviews progress is also going well and milestones are being reached), control is normal.
- But if the problems occur, project manager has to control them as quickly as possible. After the problem has been identified, additional resources may be focused on the problem areas, staff may be redeployed or the project schedule can be changed.
- If the project manager is facing with deadline pressure, experienced project managers sometime have to take the decision about the project scheduling and this control technique can be called as time-boxing.
- The time-boxing approach identifies that the whole product may not be deliverable by the predefined target. Hence, an incremental software process is taken into consideration, and a schedule is preceded for each incremental delivery.

Syllabus Topic : TimeLine Charts

3.8.1 TimeLine Charts

Q. 3.8.2 What are the TimeLine charts? (Ref. sec. 3.8.1) (5 Marks)

- Timelines are very important in project management because they help to visualize time-related activities, organization of task, set deadlines and have to define delays.
- The diagrams are useful for managers who want to get a high-level look at their tasks or to view any time-related activities.

Timeline chart helps to visualize three main timeframes :

1. **Planned time**

   Actual time in-progress that shows how long the tasks have been in progress.

2. **Forecasted time**

   - Project managers have knowledge regarding that setting expectations for delivery time is significant and challenging concern of management.
   - With the help of timelines and charts, it is possible to avoid low-quality releases as well as deadlines.
   - The diagrams add the transparency and give a chance to analyze what happened in the past and plan the future.
   - Usually, managers use timeline charts for :
     1. Projects and tasks planning
     2. Road mapping
     3. Task management

- With the help of timeline charts, users can plan many targets to be achieved on one screen and show how they can be connected with each other.
- It's possible to compare planned and actual end dates and get forecasts.

3. **TimeLine Components**

- The time line components are the diagrammatic representations of the tasks. Timelines may be highly detailed or simple. They can contain hundreds of tasks and subtasks or have only a few deliverables.
- If anyone want to build a timeline to their goals, they have to pay the attention on following :
  1. The set of tasks and objectives to be completed;
  2. Approved dates and deadlines,
  3. Dependencies between tasks,
  4. Expected duration of tasks.
- Timelines are in many forms, but the most popular and powerful option is Gantt Chart.
- Gantt Chart includes horizontal bars which represent the duration of tasks. It really provides an easy visual depiction of the project.

☞ **Purpose for the TimeLine Charts**

- The main purpose is to so make sure everything is planned properly and runs according to given schedule.

- Timeline charts considered as better for projects having many people. It doesn't insist for lot of time or much effort to form a chart.
- Online project timelines are same as of the Gantt charts as these both diagram types are concerned with time and events.

Syllabus Topic : Earned Value Analysis

3.8.2 Earned Value Analysis

Q. 3.8.3 What is Earned Value Analysis in Project tracking concept? (Ref. sec. 3.8.2) (5 Marks)

- Earned Value Analysis (EVA) is the key technique used in Project Management.
- The purpose of Earned Value Analysis is to understand how the project is progressing.
- EVA used to estimate the progress of a project based on earnings or money and schedules are calculated on the basis of EVA.
- In simple words Earned Value is a measure of 'Progress' to evaluate 'Percentage of Completeness'

3.8.2(A) Features of EVA

- Earned Value Analysis's objective is to measure project performance in terms of scope, cost and time.
- EVA is used to evaluate project health and project performance.
- Earned Value Analysis is also used for monitoring the progress of a software project / team which involves tasks allocated to the Project Schedule.
- Total time required to complete the project is calculated and each task is given an Earned Value, based on its estimated (%) out of the total.

3.8.2(B) Need for EVA

- EVA provides the measures of project process of different tasks associated with project, so it is a single way of measuring each and every point in the project.
- It also provides a signal for corrective action in the project. The types of signals can be the following :

1. Time to recover

If the project is not going as per plan and it is found that there is some delay in the project. Then at this time, situation is needed to be taken care by finding out the reasons that are causing delay and by taking the required corrective actions.

2. Request for additional funds/resources

While there is time to recover, the need for extra resources or money can be calculated with an early warning.

Key Elements of EVA

1. **Planned Value (PV)**

   The allocated cost for the project which is approved. It was known as Budgeted Cost of Work Scheduled (BCWS).

2. **Earned Value (EV)**

   The budgeted value of the completed work packages. It used to be known as Budgeted Cost of Work Performance at a specified point (BCWP).

3. **Actual Cost (AC)**

   The actual cost involved during the execution of project work. It was previously called Actual Cost of Work Performed (ACWP).

Tools and techniques

There are several software packages available which are listed are as follows :

1. Schedule maker
2. Planisware OPX2
3. RiskTrak
4. Winsight
5. Primavera

Earned Value Analysis – Example

- Consider you are handling a software development project. The project deadline of the project is eight months with costing of $10,000 per month.
- After two months, you get understood that the project work is thirty percent completed with costing of $40,000. Now there is need to determine whether the status of project will be on-time and on-budget after two months.

Step 1 : Calculate the Planned Value (PV) and Earned Value (EV)

From the scenario,

1. Budget at Completion (BAC) = $10,000 * 8 = $80,000
2. Actual Cost (AC) = $40,000
3. Planned Completion = 2/8 = 25%
4. Actual Completion = 30%

Therefore,

Planned Value = Planned Completion (%) * BAC = 25% * $ 80,000 = $ 20,000

Earned Value = Actual Completion (%) * BAC = 30% * $ 80,000 = $ 24,000

Step 2 : Compute the Cost Performance Index (CPI) and Schedule Performance Index (SPI)

Cost Performance Index (CPI) = EV / AC = $24,000 / $40,000 = 0.6

Schedule Performance Index (SPI) = EV / PV = $24,000 / $20,000 = 1.2

64

- **Interpretation :** Since Cost Performance Index (CPI) is less than one, this means the project is over budget. For every dollar spent we are getting 60 cents' worth of performance.
- Because SPI (Schedule Performance Index) is more than one, the project is considered as ahead of schedule.
- However, this gets achieved at a cost of crossing the budget. If work is continued at the current rate, the project will be definitely delivered before deadline but it will be over budget. Hence there is need of corrective measures.
- Besides computing the CPI and SPI, it is possible to calculate the earned value cost and schedule variance. Additionally, earned value forecasting formula.

Chapter Ends

CHAPTER 4

Module 4

Software Design

Syllabus

Design Principles, Design Concepts, Effective Modular Design - Cohesion and Coupling, Architectural Design Component-level design, User Interface Design.

4.1 Software Design

Q. 4.1.1 Write note on Software Design. (Ref. sec. 4.1) **(5 Marks)**

- Once the requirements document regarding the software to be developed is presented, the phase of software design gets started.
- The requirement specification activity is considered purely related with the problem domain whereas design is considered as the initial phase of transforming the problem into a solution.
- In the design phase, all the relevant entities such as the customer, business requirements and technical considerations collaborate to formulate a product or a system.
- In design process there are several elements such as set of principles, concepts and practices, which help a software engineer to model the system or product which is to be built.
- The design model is assessed for quality and reviewed before generation of code and execution of tests.
- The design model gives detailed information regarding software data structures, architecture, interfaces and components which are necessary to employ the system.

Basic of Software Design

- Software design is considered as a phase in software engineering which develops a blueprint that will be a base for constructing the software system.
- **IEEE** defines software design as 'both a process of defining the architecture, components, interfaces, and other characteristics of a system or component and the result of that process.'
- In the design phase, important and strategic decisions are made to get the expected functionality and quality of the system.
- These decisions are considered to successfully develop the software and handle its maintenance in a way that the quality of the end product will be improved.

- Software design encompasses the set of principles, concepts, and practices that lead to development of a high-quality system or product.
- Design concepts must be understood before the design practices are applied.
- Design practice itself leads to the creation of various representations of the software.
- Design Practices serve as a guide for the construction activity that follows.
- The design model provides details about software data structures, architecture, interfaces, components that are necessary to implement the system.
- Mitch Kapor, the creator of Lotus 1-2-3, presented a "software design manifesto" in Dr. Dobb's Journal.
- He said :! Good software design should exhibit :

A. Firmness : A program should not have any bugs that inhibit its function.

B. Commodity : A program should be suitable for the purposes for which it was intended.

C. Delight : The experience of using the program should be pleasurable one.

☞ **Software design model consists of 4 designs**

1. Data/class design
2. Architectural design
3. Interface design
4. Component design

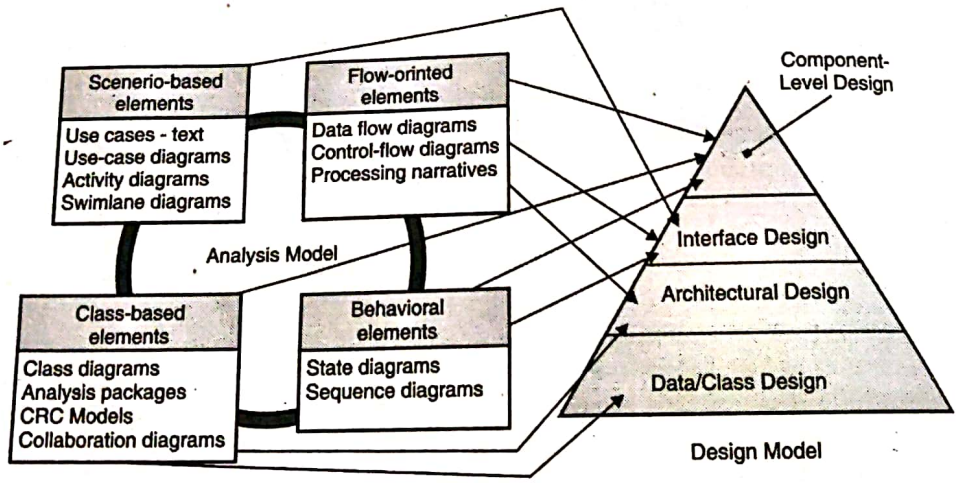


Fig. 4.1.1 : Translating Requirement Model into Design Model

☞ **Design Guidelines**

1. A design should exhibit an architecture that :
   (a) Has been created using recognizable architectural styles or patterns,
   (b) Is composed of components that exhibit good design characteristics and
   (c) Can be implemented in an evolutionary fashion!
2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems.

3. A design should contain distinct representations of data, architecture, interfaces, and components.
4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be derived using a repeatable method that is driven from information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.

4.1.1 Qualities of Good Design

Q. 4.1.2 Explain qualities of good software design. (Ref. sec. 4.1.1) **(5 Marks)**

There are number of qualities of good design :

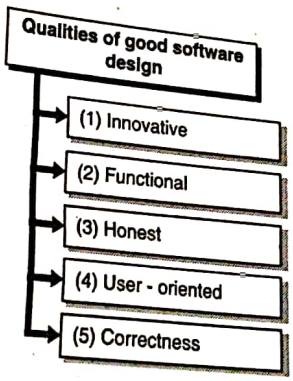


Fig. 4.1.2 : Qualities of good software design

➜ **(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.

Syllabus Topic : Design Principles

4.2 Design Principles

Q. 4.2.1 What are the design principles ? (Ref. sec. 4.2) **(5 Marks)**

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.

Syllabus Topic : Design Concepts

4.3 Design Concepts

Q. 4.3.1 With reference to software design give the meanings of
(i) Modularity (ii) Information hiding (Ref. sec. 4.3) **(5 Marks)**

- All of the software processes are characterized by basic concepts in conjunction with some practices or methods.

- Methods represent the way by which the concepts are applied. An Old technology is replaced by new technology, several modifications occur in the methods which are used to apply the concepts for the process of developing the software.
- However, the fundamental concepts which underline the software design process always remain the same. These concepts are :

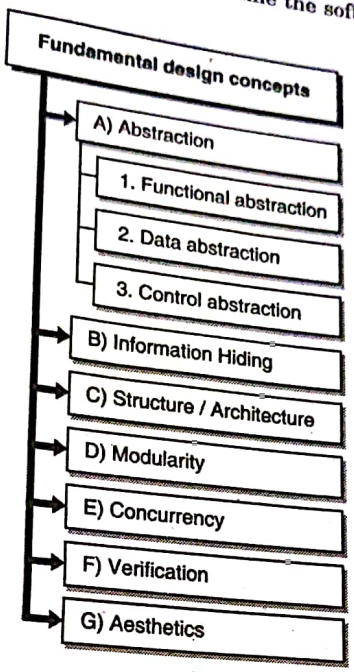


Fig. 4.3.1 : Fundamental design concepts

(A) Abstraction

Q. 4.3.2 Explain following with reference to design concepts in design modelling. : Abstraction. (Ref. sec. 4.3(A)) **(5 Marks)**

- Abstraction refers to a process by which software designers consider components at an abstract level, at the same time as ignoring the implementation details of the components.
- **IEEE** defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.'
- There are two ways to use the concept of abstraction; as a process and as an entity.
- As a **process,** abstraction refers to a mechanism of hiding irrelevant extra details and representing just the important essential features of an element so that one can concentrate on essential things at a time.
- As an **entity,** abstraction refers to a model or view of an item.
- In the software process all the steps are accomplished via several levels of abstraction.
- At the highest level, one can get the outline of the solution to the respective problem while at the lower levels, he/she will get the solution to the problem in detail.

- For example, in the requirements analysis phase, language is used to provide a solution to the problem and as one proceeds during the software process, the abstraction level reduces and source code of the software is generated at the lowest level.
- In general three abstraction mechanisms used in software design are functional abstraction, data abstraction and control abstraction.
- These mechanisms help to control the complexity of the design process by starting from the abstract design model and ending at concrete design model in a systematic manner.

1. **Functional abstraction :** This type of abstraction includes the use of parameterized subprograms. Functional abstraction can be created as a set of subprograms known as 'groups'. In these sets, routines are exist which may be visible or invisible. The use of visible routines can be done within the containing groups and also within other groups, while invisible routines are hidden from other groups and are allowed to be used in the containing group only.
2. **Data abstraction :** This type of abstraction includes describing data which specifies a data object. For example, the data object *flower* encompasses a set of attributes (color, fragrance) which describe the flower object clearly. In this abstraction mechanism, representation as well as manipulation details are ignored.
3. **Control abstraction :** This type of abstraction states the desired effect, without conducting the exact mechanism of control. For example, in programming language like C, *if* and *while* statements are abstractions of machine code implementations containing conditional instructions.

➜ (B) Information Hiding

- It is important to specify and design modules in such a manner that the data structures as well as processing details of one module should not be accessible to any other modules.
- Only required information is transferred between the modules. The way of hiding unnecessary details is referred to as **information hiding.**
- **IEEE** defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to other modules in the system.
- Information hiding is an important aspect when there is need of modifications during the testing and maintenance phase.

☞ *Advantages of information hiding*

- *Results in low coupling.*
- *Put emphasis on communication by controlled interfaces.*
- *Reduces the possibility of adverse effects.*
- *Controls the impacts of changes in one component to others.*
- *Results in higher quality software.*

(C) Structure / Architecture

- Software architecture refers to the structure of the system, which consists of several components regarding a program/ system, the attributes (properties) of those components and the relationship among them.
- The software architecture helps the software engineers in the process of analyzing the software design proficiently.
- Additionally it also helps the software engineers in decision-making as well as handling risks. Following tasks are implemented by the software architecture :
  - Give an insight to all the relevant stakeholders which helps them to communicate with each other.
  - Spot the early design decisions having big impact on the software engineering activities such as coding and testing which are followed after the design phase.
  - Generates intellectual models regarding the way by which the system is organized into components and the way by which these components communicate with each other.

(D) Modularity

Q. 4.3.3 Explain modularity. (Ref. sec. 4.3(D)) **(10 Marks)**

- Modularity can be achieved by the process of dividing the software into components which are uniquely named and addressable. These components are also known as **modules.**
- A complex system (huge application) is partitioned into a set of discrete modules in such a manner that one should be able to develop those modules independent of each other.
- After development is completed, the modules are integrated to form an entire project.
- One has to remember that as the number of modules a system is divided into increases, it becomes easy to handle the software but it also increases the effort required to integrate the modules.
- The process of modularizing a design helps to plan the development in a more efficient way, accommodate changes without difficulty, conduct testing and debugging effectively as well as efficiently, and carry out the maintenance work without adversely affecting the functionality of the software.

Example

- The ubiquitous television set is an example of a system made up of a number of modules – speakers, projection tube, channel buttons, volume buttons, etc.
- Each module has its own defined functionality but when they are put together synergistically, the complete functionalities of a television are realized.

(E) Concurrency

- It is important to utilize the resources efficiently as much as possible. For this purpose multiple tasks must be executed concurrently.

- This aspect makes concurrency one of the important concepts of software design.
- Every system should be designed in such a manner that it should facilitate multiple processes to execute concurrently.
- For example, if the currently executing process is waiting for some resource, the system must be able to execute any other process in the mean time.

➜ **(F) Verification**

- Verification is described as a mechanism of confirmation by examining and giving evidence that there is no mismatch in design output and the design input specifications.
- Design input can be any physical as well as performance requirement which is used as the basis for the purpose of designing.
- Design output is considered as the result of all the design phases' efforts. The final design output is a basis for device master record.

➜ **(G) Aesthetics**

- Aesthetics is the philosophical study of beauty and taste.
- Software engineering is a largely creative design discipline, where new designs (both graphical design models as well as code) are created from scratch.
- Compared to art, there are many targets (e.g. desired functionality) and constraints (e.g. desired quality attributes) that the software engineer must adhere to, but these still leave almost infinitely many ways to create a model that fulfills all those requirements.
- In addition, for essentially the same model, there are many ways to represent it. Since models and code must also serve to communicate ideas to other people (or to the same people months or years later), the chosen structure and representation are key attributes for the success of the product.
- It's a well-accepted hypothesis that the **aesthetics** of the artifacts produced during software development are an indication of 'the quality': it is very common to comment on designs as being 'beautiful" or "elegant".
- Such a qualification is used to indicate the apparent quality of the design. Apparently, we make an intuitive connection between aesthetics and quality.

Syllabus Topic : Effective Modular Design

4.4 Effective Modular Design

Q. 4.4.1 Write note on effective modular design. (Ref. sec. 4.4) **(10 Marks)**

- Modularity can be defined as dividing the software into distinctively named and addressable components, which are also called as modules.
- A complex system (large program) is divided into a set of distinct modules in such a way that each module can be developed independent of other modules.

- After developing the individual modules, all these modules are incorporated together to fulfill the software requirements.
- As the modules are developed separately if the number of modules is very huge then it requires more efforts for incorporating them.

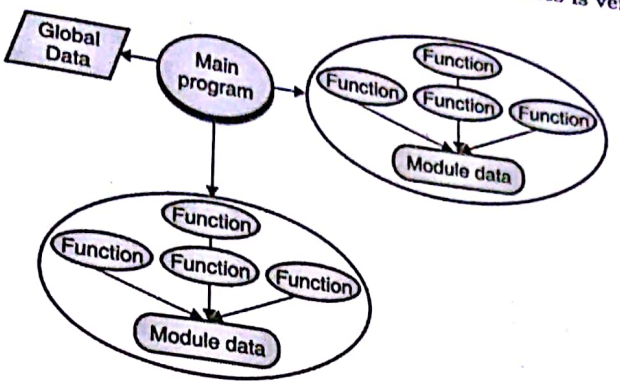


Fig. 4.4.1 : Modularity

- Modularizing a design assists to plan the development in a more efficient manner, put up changes easily, carries testing and debugging effectively and efficiently, and conducts maintenance work without harmfully affecting the functioning of the software.

4.4.1 Advantages of Modularization

1. Maintaining smaller components is very easy.
2. According to the functional aspects the program can be divided.
3. As per the requirement, the level of abstraction can be carried out in the program.
4. Components with high cohesion can be re-used again.
5. Concurrent execution can be made possible.
6. Security can be achieved.

Modularity has become an accepted approach in all engineering disciplines. A modular design reduces complexity, facilitates change (a critical aspect of software maintainability), and results in easier implementation by encouraging parallel development of different parts of a system.

4.4.2 Functional Independence

Q. 4.4.2 Write note on function independence. (Ref. sec. 4.4.2) (5 Marks)

- The concept of functional independence is considered as a direct outcome of the modularity and the concepts of abstraction as well as information hiding.

- Functional independence is implemented by the process of developing modules with "single-minded" function and an "a version" with extreme communication with other modules.
- In other words, we need to design software in order that each module is concerned with a particular sub-function of requirements and posses only one interface when seen from different parts of the program structure.
- One good question is that why independence is important. Software having efficient modularity, that is, independent modules, is simple to develop since function may be compartmentalized and simplification of interfaces is done (consider the process of ramifications when a team conducts development).
- It is simple to maintain independent modules (and test) since there is limitation in secondary effects which are generated by design or code modification. Also there is reduction in error propagation, and possible to reuse modules.
- As a result functional independence is a key to good design, and in turn the respective design is the key to good quality of the software.
- There are two qualitative criteria to measure independence : cohesion and coupling.
- Cohesion is used to measure the relative functional strength of a module where as coupling is used to measure the relative interdependence in between modules.

Syllabus Topic : Cohesion

4.5 Cohesion

→ **(MU - May 15, May 16, Dec. 16, May 17, Dec. 17, May 18)**

Q. 4.5.1 What is cohesion? Explain different forms of it. (Ref. sec. 4.5)

May 15, May 16, Dec. 16, May 17, Dec. 17, May 18, 5 Marks

- 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.

4.5.1 Different Types of Cohesion

Q. 4.5.2 Explain different types of cohesion. (Ref. sec. 4.5.1)

→ **(MU - Dec. 15)**

Dec. 15, 5 Marks

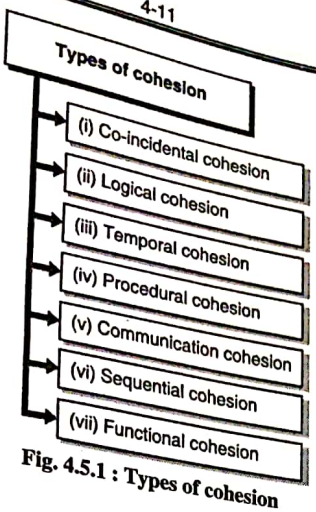


Fig. 4.5.1 : Types of cohesion

(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.

4.5.2 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.

4.5.3 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.

Syllabus Topic : Coupling

4.6 Coupling

➜ **(MU - May 15, May 16, Dec. 16, May 17, Dec. 17, May 18)**

Q. 4.6.1 What is coupling? Explain different forms of it. (Ref. sec. 4.6) **May 15, May 16, Dec. 16, May 17, Dec. 17, May 18, 5 Marks**

- 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,

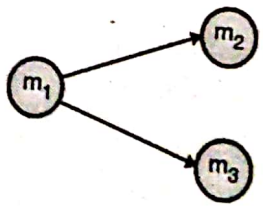


Fig. 4.6.1 : Coupling

- 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.

4.6.1 Types of Coupling

Q. 4.6.2 Explain different types of coupling. (Ref. sec. 4.6.1) → (MU - Dec. 15) **Dec. 15, 5 Marks**

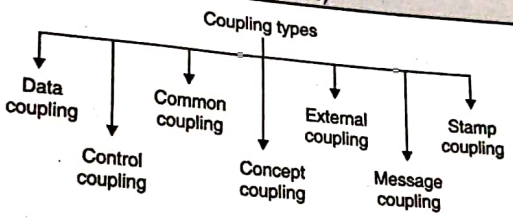


Fig. 4.6.2 : Coupling types

(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.

4.6.2 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.

4.6.3 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.

4.6.4 Advantages of High Cohesion and Low Coupling

Q. 4.6.3 What are benefits of high cohesion and low coupling ? (Ref. sec. 4.6.4) → (MU - Dec. 17)
Dec. 17, 4 Marks

☞ **Benefits of high cohesion**

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

☞ **Benefits of low coupling**

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

4.7 Differences between Coupling and Cohesion

Q. 4.7.1 Differentiate between cohesion and coupling. (Ref. sec. 4.7) **(5 Marks)**

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.

Syllabus Topic : Architectural Design

4.8 Architectural Design

Q. 4.8.1 Explain architectural design. (Ref. sec. 4.8) **(10 Marks)**

- Requirements of the software should be transformed into an architecture that describes the software's top-level structure and recognizes its components.
- This can be achieved by using the architectural design which acts as an initial 'blueprint' from which software can be developed.
- IEEE described architectural design as a process of defining a set of hardware and software components and their interfaces to set up the framework for the development of a computer system. This framework is established by observing the software requirements document and designing a model for providing implementation details.
- These details are used to state the components of the system together with their inputs, outputs, functions, and the interaction between them.
- Architectural design can be represented with the help of following models :

1. **Structural model :** Demonstrate architecture as an ordered collection of program components.
2. **Dynamic model :** This model illustrates the behavioural aspect of the software architecture and shows how the structure or system configuration changes as the function changes because of the change in the external environment.
3. **Process model :** This model concentrates on the design of the business or technical process, which must be implemented in the system.
4. **Functional model :** This model signifies the functional hierarchy of a system.

5. **Framework model :** This model ties to recognize repeatable architectural design patterns which are found in same types of applications. This will results in increase in the level of abstraction.

4.8.1 Functions of Architectural Design

1. It describe an abstraction level at which the designers can state the functional and performance behavior of the system.
2. It estimates all top-level designs.
3. It acts as a guideline for improving the system by illustrating those features of the system that can be changed easily while maintaining the system integrity.
4. It develops and documents top-level design for the external and internal interfaces.
5. It develops initial versions of user documentation.

4.8.2 Architectural Design Document

- The result of architectural design process is Architectural Design Document (ADD).
- This document contains a number of graphical representations that encompasses software models along with related descriptive text.
- The software models consist of static model, interface model, relationship model, and dynamic process model which illustrates how the system is arranged into a process during run-time.
- Architectural design document provides the developers a solution to the problem defined in the Software Requirements Specification (SRS).
- In addition to ADD, other outputs of the architectural design are given below :

1. A range of reports including audit report, progress report, and configuration status accounts report.
2. Different plans for detailed design phase, which consist of following :
   (i) Software verification and validation plan,
   (ii) Software configuration management plan,
   (iii) Software quality assurance plan,
   (iv) Software project management plan.

4.8.3 Architectural Style

Q. 4.8.2 Explain different Architectural styles with suitable brief example of each. (Ref. sec. 4.8.3) → (MU - Dec. 15)

Dec. 15, 10 Marks

- 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 :

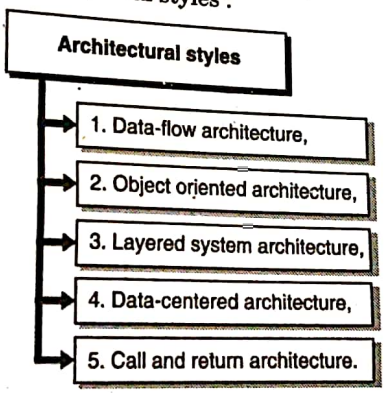


Fig. 4.8.1 : Architectural styles

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.

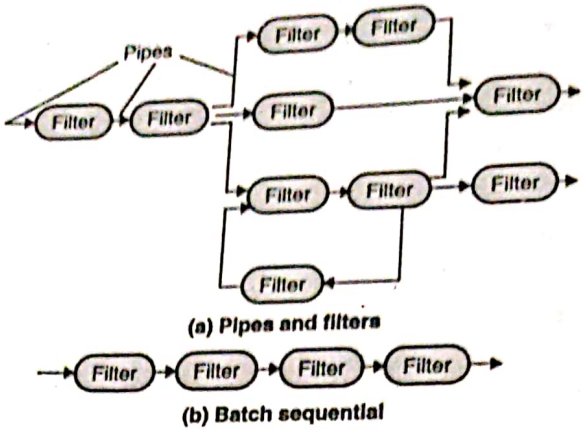


Fig. 4.8.2 : Data flow architecture

- 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.

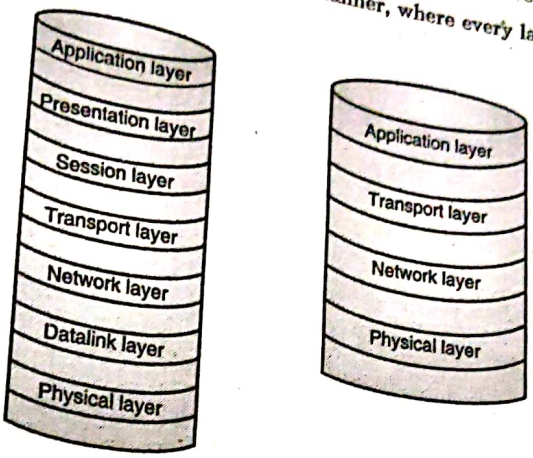


Fig. 4.8.3 : OSI and Internet Protocol Suite

- 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.

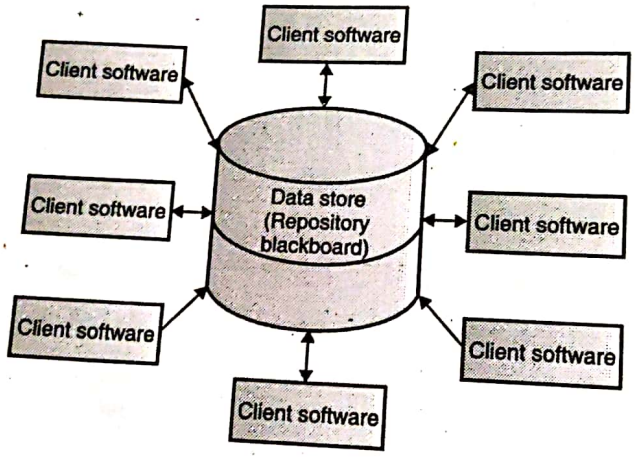


Fig. 4.8.4 : Data-centered architecture

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.

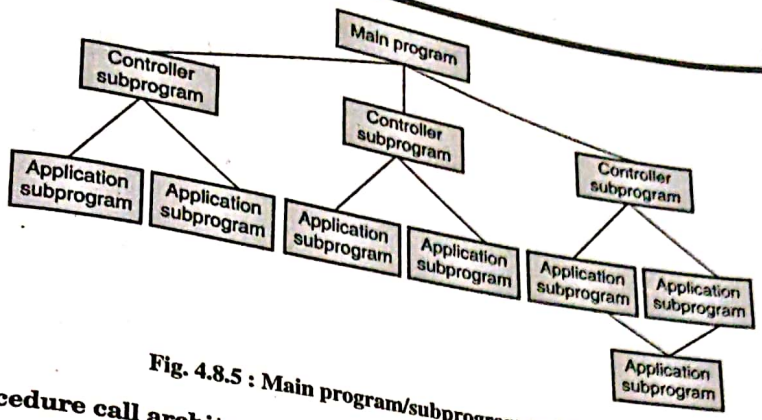


Fig. 4.8.5 : Main program/subprogram architecture

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.

Syllabus Topic : Component - Level Design

4.9 Component - Level Design

Q. 4.9.1 Write note on component level design. (Ref. sec. 4.9) (10 Marks)

- The main function of component level design is to define data structures, algorithms, interface characteristics as well as communication mechanisms for all the present software components which are identified in the phase of architectural design.
- The phase of component level design is expected when the data, architectural, and interface designs are completed.
- The software is represented by the component-level design in such a manner which lets the designer to review it for correctness as well as consistency prior to its built.
- The work product generated is considered as a design for all of the software components. It is represented with the help of graphical, tabular, or text-based notation.
- Design walkthroughs are carried out to observe accuracy of the data transformation or control transformation which has been allocated to all of the components in the earlier design steps.

Component Definitions

- A **component** is a modular, deployable, replaceable part of a system which encapsulates implementation and exposes a set of interfaces.
- As per object-oriented view a component contains a set of collaborating classes
- As per traditional view, a component (or module) resides in the software and serves one of three roles as follows :

- **Control components** coordinate invocation of all other problem domain components
- **Problem domain components** implement the functionality needed to the customer
- **Infrastructure components** implement the functionality required to support the processing needed in a domain application.

– Process-Related view focus on building systems out of existing components which are selected from a catalog of reusable components as a way of populating the architecture.

☞ **Object-Oriented Component Design**

– Put emphasis on describing the domain oriented analysis classes as well as the definition of infrastructure classes.

– Comprehensive information of class attributes, operations, and interfaces is necessary before beginning construction activities.

☞ **Principles for Designing Class-based Components**

– *Open-Closed Principle (OCP)* : Extensions should be allowed in the class but not modification.

– *Liskov Substitution Principle (LSP)* : It should be possible to substitute subclasses for their base classes.

– *Dependency Inversion Principle (DIP)* : Depend on abstractions, do not depend on concretions.

– *Interface Segregation Principle (ISP)* : Multiple client oriented interfaces are considered as better compared to one general purpose interface.

– *Release Reuse Equivalency Principle (REP)* : The granule of reuse is the granule of release.

– *Common Closure Principle (CCP)* : Classes that change together belong together.

– *Common Reuse Principle (CRP)* : Classes that can't be used together should not be grouped together.

☞ **Component-Level Design Guidelines**

1. Components

– Establish naming conventions in the phase of architectural modeling.

– Names of Architectural components should be meaningful for the stakeholders.

– Names of Infrastructure components must have implementation specific meanings.

– The nature of components can be identified with the help of UML stereotypes.

2. Interfaces

– Use of lollipop representation is considered as better as compared to formal UML box and arrow notation.

– For the purpose of consistency, the flow of interfaces should be from the left-hand side of the component box.

– Display only those interfaces which are relevant to the component under construction.

3. **Dependencies**
   - To improve readability it should be better to have the model dependencies from left to right and inheritance from bottom (child classes) to top (parent classes).
   - Interfaces must be used to represent component interdependencies rather than component to component dependencies.

- **Cohesion**
  - *Utility cohesion* : Components are combined in a group within the same category but are otherwise unrelated.
  - *Temporal cohesion* : Operations are carried out to reflect a specific behavior or state.
  - *Procedural cohesion* : Components are combined in a group to let one be called immediately after the preceding one was called with or without passing data.
  - *Communicational cohesion* : Operations need same data are combined in a group in same class.
  - *Sequential cohesion* : Components combined in a group to allow input to be passed from first to next and so on.
  - *Layer cohesion* : Exhibited by the package components when lower layer's service is accessed by a higher level layer, but higher level layer's services are not accessed by lower layer.
  - *Functional cohesion* : Module performs one and only one function.

- **Coupling**
  - *Data coupling* : This coupling takes place once long strings of arguments are passed between components.
  - *Stamp coupling* : This coupling takes place once parts of bigger data structures are passed between components.
  - *Control coupling* : This coupling takes place once control flags are passed as arguments by one component to another.
  - *External coupling* : This coupling takes place once a component communicates or collaborates with any 7 of the infrastructure components (e.g., database).
  - *Common coupling* : This coupling takes place once a global variable is used by multiple components.
  - *Content coupling* : This coupling takes place once one component secretly makes changes in internal data of another component.
  - *Routine call coupling* : This coupling takes place once one operator is invoked by another component.
  - *Type use coupling* : This coupling takes place when a data type defined in one component is used by another component.
  - *Inclusion or import coupling* : This coupling takes place when one component imports a package or uses the content of another component.

Conducting Component Level Design

1. Identify all of the respective design classes which are corresponding to the problem domain.
2. Identify all of the respective design classes which are corresponding to the infrastructure domain.
3. Elaborate all of the design classes which are not acquired as reusable components.
   a. Specify details regarding the message when there is collaboration of classes or components.
   b. Identify suitable interfaces for all the components.
   c. Elaborate attributes and define data types and data structures which are necessary for their implementation.
   d. Describe processing flow of all the operations thoroughly.
4. Make the Identification of persistent data sources such as databases and files and recognize the classes which are necessary to manage them.
5. Develop and elaborate behavioral representations for all the classes or components.
6. Elaborate deployment diagrams for the purpose of providing additional implementation details.
7. Factor all of the component-level diagram representations and consider alternatives.

Object Constraint Language (OCL)

- Complements UML by permitting a developer to use formal grammar as well as syntax for the purpose of constructing unambiguous statements regarding design model element.
- Parts of OCL language statements :
  1. A context which indicates the restricted situation in which the statement is considered as valid.
  2. A property which represents few characteristics of the context.
  3. An operation which manipulates or qualifies a property.
  4. Keywords which are used for the purpose of specifying conditional expressions.

Conventional or Structured Component Design

- Each block of code has a single entry at the top.
- Each block of code has a single exit at the bottom.
- Only three control structures are required : sequence, condition (if-then-else), and repetition (looping).
- Reduces program complexity by enhancing readability, testability, and maintainability.

Design Notation

- *Flowcharts* : Arrows for flow of control, diamonds for decisions, rectangles for processes.
- *Decision table* : Subsets of system conditions and actions are associated with each other to define the rules for processing inputs and events.

- *Program Design Language (PDL)* : Structured English or pseudocode used to describe processing details.

Program Design Language Characteristics

- Predetermined syntax with keywords provided for the purpose of representing all structured constructs, data declarations as well as module definitions.
- Free syntax of natural language for the purpose of describing processing features.
- Data declaration facilities for simple as well complex data structures.
- Subprogram definition as well as invocation facilities.

Design Notation Assessment Criteria

- *Modularity* : Notation provide support for development of modular software.
- *Overall simplicity* : Easy to learn, easy to use, easy to write.
- *Ease of editing* : Easy to make changes in design representation whenever required.
- *Machine readability* : Notation can be input directly into a computer-based development system.
- *Maintainability* : Maintenance of the configuration in general engage maintenance of the procedural design representation.
- *Structure enforcement* : Enforces the use of structured programming constructs.
- *Automatic processing* : Allows the designer to verify the correctness and quality of the design.
- Data representation : Ability to represent local as well as global data directly.
- *Logic verification* : Automatic logic verification improves testing competence.
- Easily converted to program source code (makes code generation quicker).

Syllabus Topic : User Interface Design

4.10 User Interface Design

→ **(MU - Dec. 16, May 18)**

Q. 4.10.1	What is user interface design process ? (Ref. sec. 4.10)	Dec. 16, 10 Marks
Q. 4.10.2	Write short note on User Interface Design. (Ref. sec. 4.10)	May 18, 10 Marks

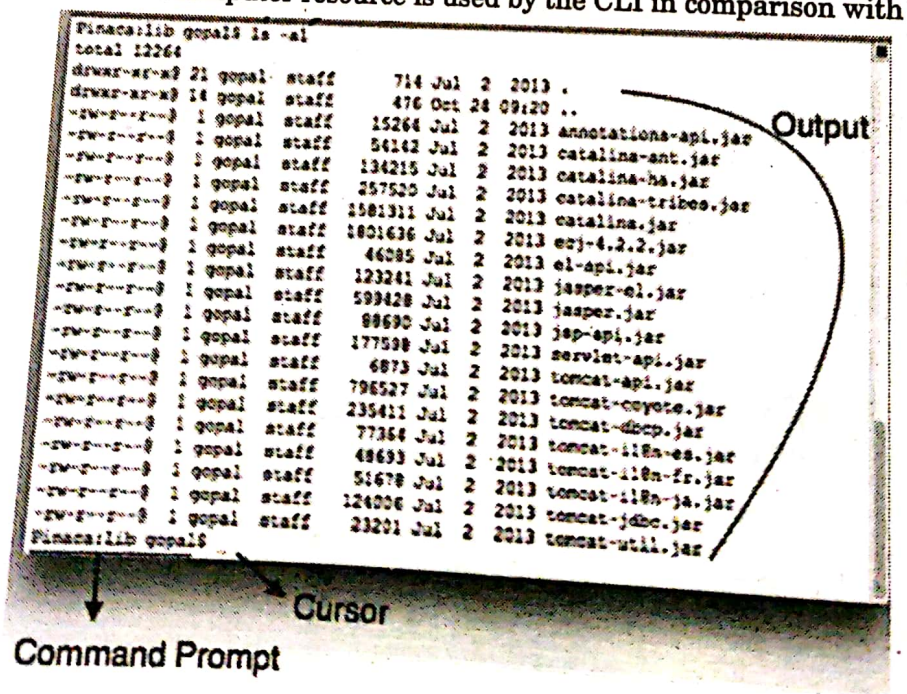
- User interface is considered as a front-end application view which is interacted by the user so as to use the software.
- User is able to manipulate as well as control the software and hardware with the help of user interface.
- Now-a-day, user interface appears in each and every place where there 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 :
  - Attractive
  - Simple to use
  - Responsive in short time
  - Clear to understand
  - Consistent on all interfacing screens
- UI is broadly divided into two categories :
  1. Command Line Interface
  2. Graphical User Interface

4.10.1 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.

CLI Elements



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

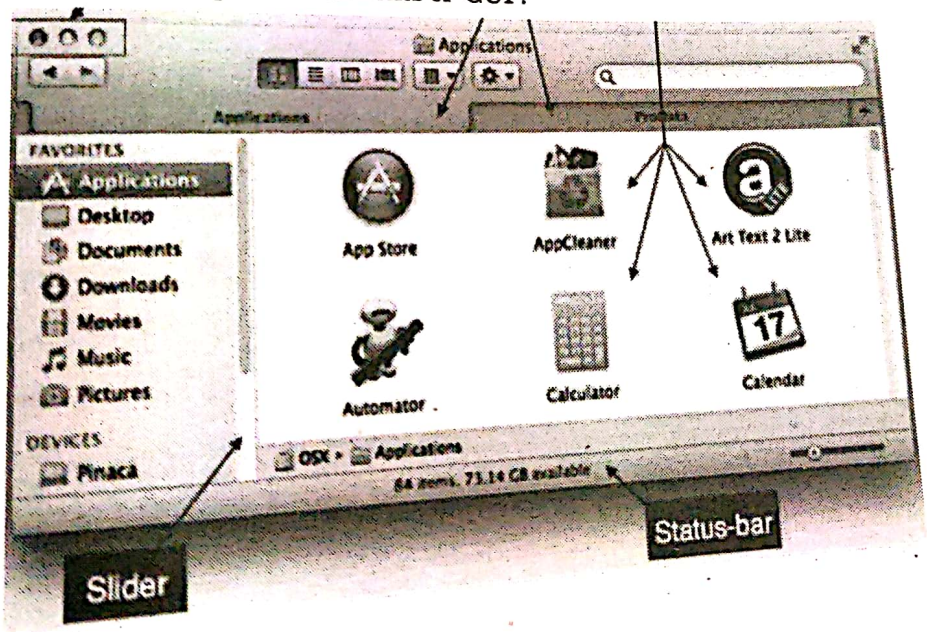
- **Command Prompt :** It is text-based notifier which usually displays the context where the user is working. Software system generates it.
- **Cursor :** It is a small horizontal line or a vertical bar with the height of line. It is used to represent position of character at the time of typing. Cursor is usefully found in blinking state. It moves when user writes or deletes text.
- **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.

4.10.2 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.

4.10.2(A) GUI Elements

- GUI provides several components for the purpose of interacting with software or hardware.
- All of these graphical components give ways to interact with the system.
- Following are some important elements of GUI :



- **Window :** It is the area in which contents of application are displayed.

- To represent file structure, the Window contents can be displayed in the form of icons or lists.
- In an exploring window, the navigation is very easy for user.

- Windows provide facility to minimize, resize or maximize. It is possible to move them anywhere on the screen. A window may contain another window (child window) of the same application inside it.
  - **Tabs :** When an application executes more than one instances of itself, then those instances appear on the screen as separate windows.
  - **Tabbed Document Interface** provides the facility to open more than one document in the same window. This interface is used to view preference panel in application. Now-a-days this feature is used by almost all web-browsers.
  - **Menu :** Menu is considered as an array of standard commands, combined together and located at a prominent place (mostly top) inside the application window. It is also possible to program the menu to appear or hide on mouse clicks.
  - **Icon :** An icon is small picture which is used to represent an application. These icons are clicked or double clicked to open the application window.
  - **Cursor :** In GUI cursors are used to represent interacting devices like mouse, touch pad, digital pen, etc.
- In almost real-time, the instructions given by hardware are followed by the cursor. Cursors which are also called as pointers are used to select menus, windows and other application features.

4.10.2(B) Application Specific GUI Components

A GUI of an application contains following elements :

- **Application Window :** Most of the application windows use the constructs provided by underlying OS but some have their own customized windows to contain the contents of application.
- **Dialogue Box :** Dialog box is a child window which has message for the user and request for some action to be taken. For Example: A dialog box is displayed by an application to get confirmation from user to delete a file.
- **Text-Box :** Used to accept input from user in text format.
- **Buttons :** Used to submit inputs to the application.
- **Radio-button :** Displays list of options for selection. Only single radio button can be selected among all offered.
- **Check-box :** Its functionality is same as of the radio button but it allows multi selection.
- **List-box :** Provides list of available items using single control. Multiple items can be selected.

4.10.2(C) User Interface Design Activities

- Several activities are carried out for designing user interface. The process of GUI design and implementation is same as of the SDLC (Software Development Life Cycle).
- For GUI implementation there are number of models available such as Waterfall, Iterative or Spiral Model.

- Following GUI specific steps should be fulfilled by the model used for GUI design and development.

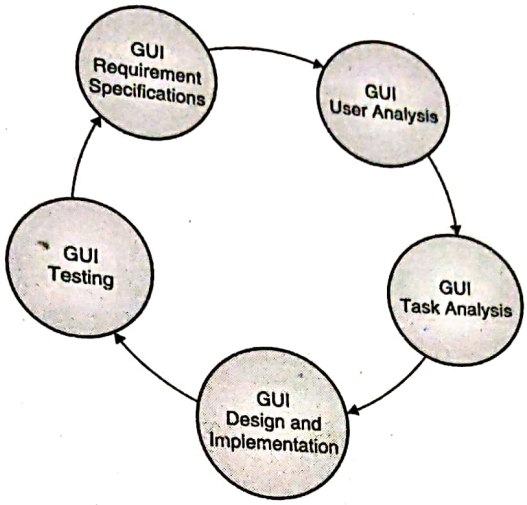


Fig. 4.10.1 : GUI Steps

- **GUI Requirement Gathering :** The designers need the list of list of all functional as well as non-functional requirements regarding GUI. These requirements are accepted from users and their existing software solution.
- **User Analysis :** The designer do the study regarding who will actually use the software GUI. The target audience is an important aspect since the design details change according to the knowledge as well as competency level of the end user.

- If user looks to be techno savvy then advanced and complex GUI can be developed.
- For a new user, more informative data is included regarding how-to of software.
  - **Task Analysis :** Designers have an important responsibility to analyze the task which is to be done by the respective software solution.
- Here in case of GUI, it is not important how it will be done. It is possible to represent the task in hierarchical manner considering one major task at the top and dividing it further into smaller sub-tasks.
- For GUI presentation, goals are provided by the tasks.
- In the software the flow of GUI contents is decided by the flow of information among sub-tasks.
  - **GUI Design and implementation :** Designers after getting complete information regarding requirements, tasks and user environment, design the respective GUI and implement the

design into code and integrate the GUI with currently working or any dummy software in the background. Then self-testing is implemented by the developers.

- **Testing :** There are several ways to implement the GUI testing. Organization has options such as in-house inspection, straight involvement of end users and prior release of beta version. Important aspects verified in testing are usability, compatibility, user acceptance etc.

4.10.2(D) GUI Implementation Tools

- There are number of tools available for designers to create entire GUI on a single mouse click.
- Out of them some tools can be easily integrated into the software environment.
- A huge set of GUI controls is provided by GUI implementation tools.
- The code is modified by the designers for software customization.
- There are diverse segments of GUI tools in accordance with their use and platform.

Example

- Mobile GUI, Computer GUI, Touch-Screen GUI etc.
- Following are some tools which are used to build GUI :
  - FLUID
  - AppInventor (Android)
  - LucidChart
  - Wavemaker
  - Visual Studio

4.10.2(E) User Interface Golden Rules

→ (MU - May 15)

Q. 4.10.3 What are the features of good user interface? (Ref. sec. 4.10) May 15, 5 Marks

Q. 4.10.4 Explain user interface golden rules. (Ref. sec. 4.10) (10 Marks)

- 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, described by Shneiderman and Plaisant in their book (Designing the User Interface).

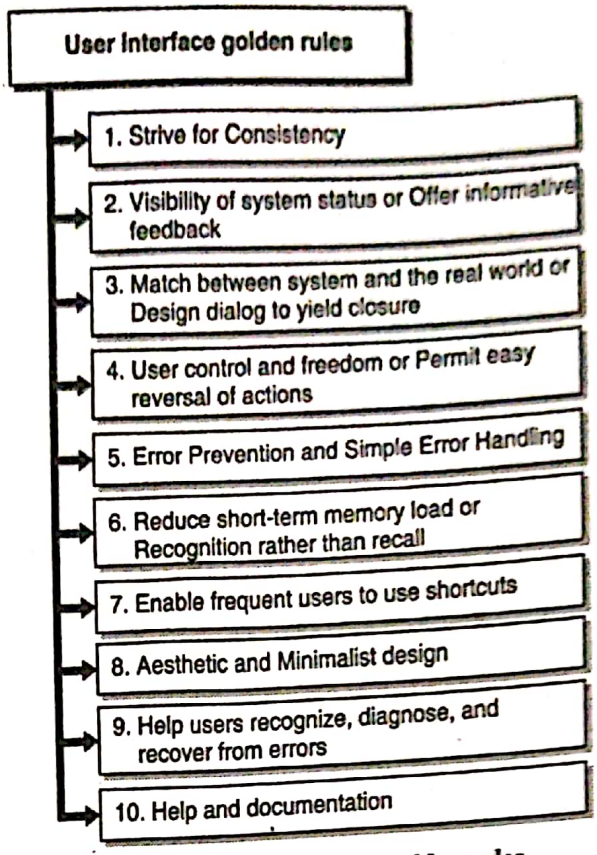


Fig. 4.10.2 : User interface golden rules

1. Strive for Consistency

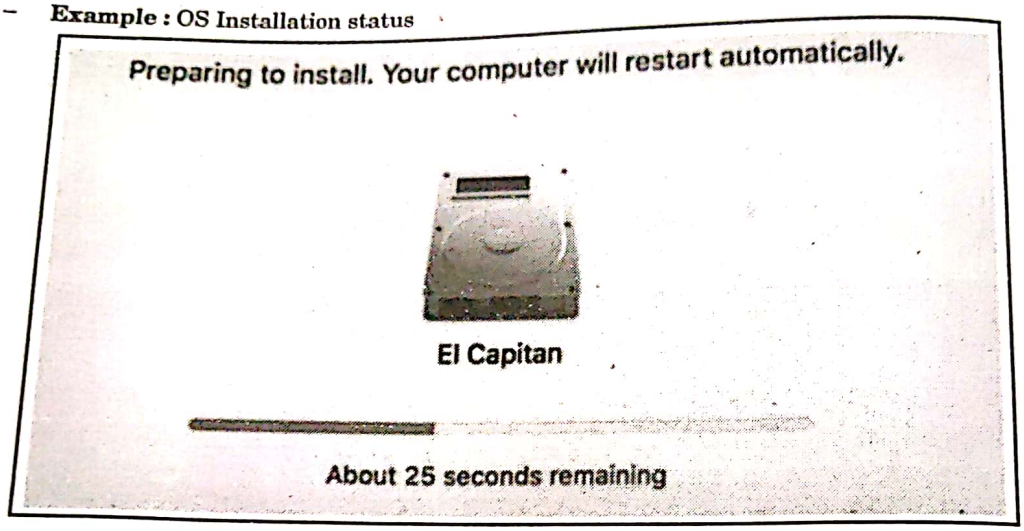
- Users should not have to wonder whether different words, situations, or actions mean the same thing. Do not confuse your user keep *words and actions* consistent.
- Use "The Principle of Least Surprise".
- **Example :** Car Climate Control

- 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 :**
  - Workflow / Processes
  - Functionality

- Appearance
- 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

- *Relevant*
- *Fits importance and urgency*
- *Comprehensible and meaningful*
- *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)*

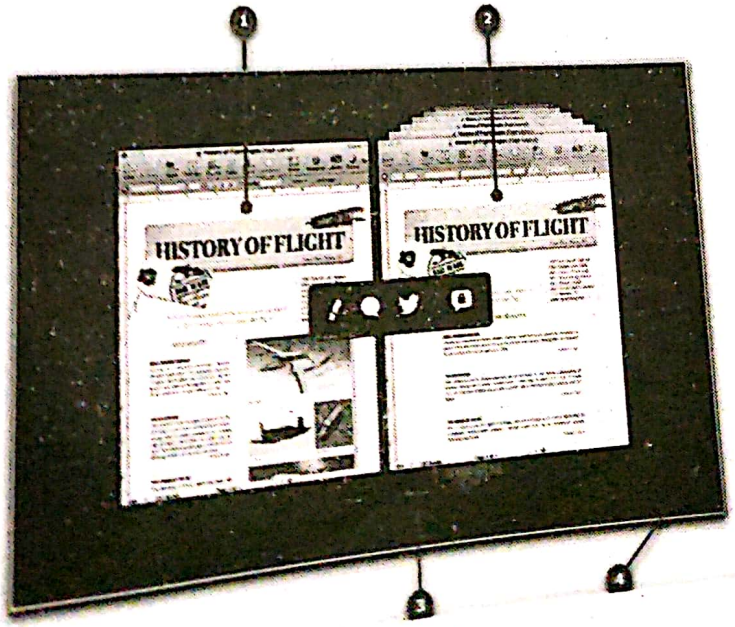
- Sequence of actions should be organized into groups with a beginning, middle and end. When a process is finished, remember to display a notification message.
- Let the user know that he has done all that's needed.

Design

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

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

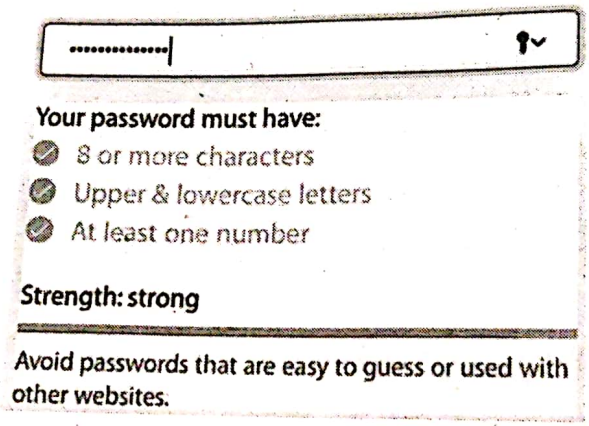
- Shneiderman puts it nicely "This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options."
- **Example :** Document History



- 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
  - No interference with workflow
  - More freedom for the user
  - 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



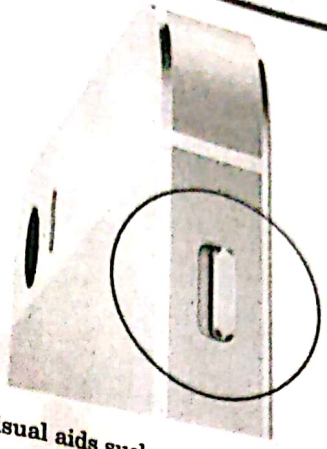
- As much as possible, design the system so the user cannot make a serious error.
- If an error is made, the system should be able to detect the error and offer a simple, comprehensive mechanism for handling the error.

Error Prevention

- Error prevention over error correction
- Automatic detection of errors
- Clear error notifications
- 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

- "Recognition over recall"
- Implicit help
- 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

New Window	⌘N
New Private Window	⇧⌘N
New Tab	⌘T
Open File...	⌘O
Open Location...	⌘L

Shortcuts

- Keyboard shortcuts
- "Power User" features
- Action automation

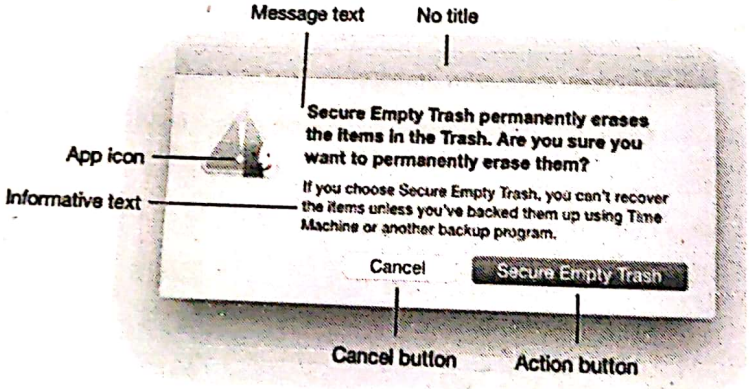
➜ **8. Aesthetic and Minimalist design**

- 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 Ends...

□□□

CHAPTER 5

Module 5

Software Risk, Configuration Management & Quality Assurance

Syllabus

Risk Identification, Risk Assessment, Risk Projection, RMMM, Software Configuration management, SCM repositories, SCM process, Software Quality Assurance Task and Plan, Metrics, Software Reliability, Formal Technical Review (FTR), Walkthrough.

5.1 Risk Management

→ **(MU - May 15, Dec. 15, Dec. 16, May 18)**

Q. 5.1.1	What is software risk ? (Ref. sec. 5.1)	May 15, 2 Marks
Q. 5.1.2	Discuss the different categories of risks. (Ref. sec. 5.1)	Dec. 15, 5 Marks
Q. 5.1.3	Explain different steps in risk management with suitable diagram. (Ref. sec. 5.1)	Dec. 15, 5 Marks
Q. 5.1.4	Write short note on Risk Management. (Ref. sec. 5.1)	Dec. 16, 10 Marks
Q. 5.1.5	Explain Risk and its types. (Ref. sec. 5.1)	May 18, 5 Marks

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.

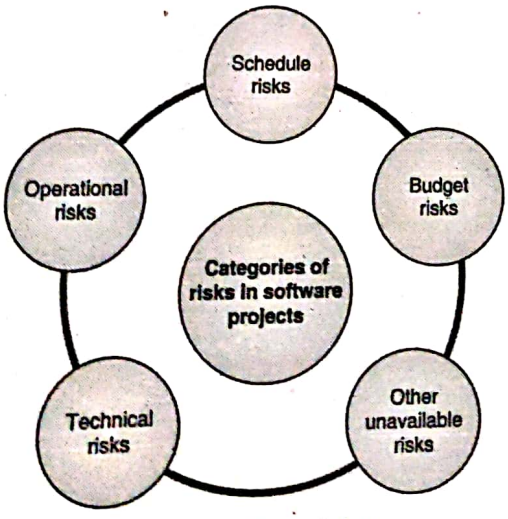


Fig. 5.1.1 : Kinds of risks

Various Kinds of Risks Associated with Software Project 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 :

- Incorrect Time Estimation, and consequently an incorrect project schedule.
- Improper Resource Allocation.
- Underutilization of Resources.
- Superficial Understanding of Project Complexities.
- 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 :

- Improper Budget Estimation.
- Cost Overruns due to underutilization of resources.
- Expansion of Project Scope.
- 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 :

- Improper Process Implementation.
- Silo approach followed by software development teams leading to conflicts.
- Conflicting Priorities.
- Lack of conflict resolution / team spirit.
- Lack of clarity in responsibilities.
- Breakdown in communications.
- 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.

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.
  - Changes in government policy,
  - Obsolescence (becoming outdated) of software due to new technology from a rival company,
  - 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 :
  - Assess continuously what can go wrong (risks 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

Syllabus Topic : Risk Assessment

5.2 Risk Assessment

Q. 5.2.1 Describe the following with respect to Risk assessment :
(i) Risk identification (ii) Risk analysis (Ref. sec. 5.2) **(5 Marks)**

In Risk Assessment, mainly following elements are there :

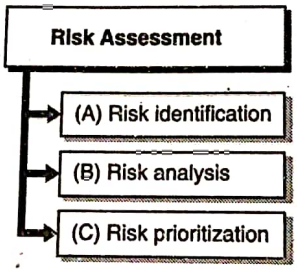


Fig. 5.2.1 : Risk assessment

➜ (A) Risk Identification

Produces lists of the project-specific risk items likely to compromise project's success.

➜ (B) Risk Analysis

Assesses the loss probability and loss magnitude for each identified item, and it assesses compound risks in risk-item interactions.

➜ (C) Risk Prioritization

Produces a ranked ordering of the risk items identified and analyzed.

Syllabus Topic : Risk Identification

5.2.1 Risk Identification

Q. 5.2.2 Explain risk identification. (Ref. sec. 5.2.1) **(5 Marks)**

- It is a process of stating which risk may occur during development of the project.
- In order to identify the risks that project may be subjected to, it is important to first study the problems faced by previous projects.
- Study the project plan properly and check for all the possible areas that are vulnerable to some or the other type of risks.
- The best way of analyzing a project plan is by converting it to a flowchart and examines all essential areas.
- It is important to conduct few brainstorming sessions to identify the known and unknowns that can affect the project.
- Any decision taken related to technical, operational, political, legal, social, internal or external factors should be evaluated properly.
- In this phase of Risk management we have to define processes that are important for risk identification.

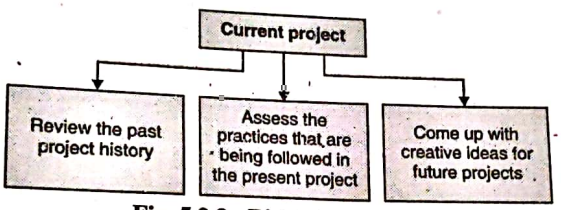


Fig. 5.2.2 : Risk identification

- All the details of the risk such as unique Id, date on which it was identified, description and so on should be clearly mentioned.
- There are in general two types of risks : generic risks and product-specific risks.
  - **Generic risks** are considered as a potential threat to each and every software project.
  - **Product-specific** risks can be identified only by those who are having a clear understanding regarding the technology, the people, and the particular environment in which software is to be built.
- For identification of product-specific risks, there is need to examine the project plan and the software statement of scope, and an answer to the given question should be found out: "Which particular characteristics of the respective product may threaten the associated project plan?"
- One mechanism to identify risks is to generate a risk item checklist. This list helps in risk identification and highlights some subset of known as well as predictable risks in the following given generic subcategories :
  - **Product size :** Risks related to the overall size of the software which is to be built or customized.

- **Business impact :** Risks related to constraints which are imposed by the management or the marketplace.
- **Stakeholder characteristics :** Risks related to the sophistication of the stakeholders and the capability of developer to interact with stakeholders in a timely manner.
- **Process definition :** Risks related to the extent to which the software process has been defined and is followed by the respective development organization.
- **Development environment :** Risks related to the availability as well as quality of the existing tools which will be used to develop the product.
- **Technology to be built :** Risks related to the complexity of the system which is to be built and the "newness" of the initiated technology which is packaged by the system.
- **Staff size and experience :** Risks related to the overall technical as well as project experience of the associated software engineers.

- There are different ways available to organize the risk item checklist.
- Questions which are relevant to all the respective topics can be answered for each and every software project.
- The answers to these questions help to calculate the impact of risk.

5.2.2 Risk Analysis

Q. 5.2.3 Explain risk analysis. (Ref. sec. 5.2.2) **(5 Marks)**

Qualitative risk analysis

- It is a procedure in which assignment of priorities to risk can be done according to their possibility of occurrence and their effect.
- It supports the managers to decrease the uncertainty level and focuses on risks which have high priority.
- Create Plan for risk management as early as possible in the project. It can impact on different factors like cost, time, scope, quality and procurement.
- The inputs of qualitative risk analysis involves :
  - Risk management plan
  - Scope baseline
  - Risk register
- Enterprise environmental factors
  - Organizational process assets
  - The output of this stage would be
  - Project document updated

Quantitative risk analysis

- It is the process of mathematically analyzing the impact of recognized risks on complete project purpose.
- To minimize uncertainty, this type of analysis is very useful for decision making.
- The input to this stage is :
  - Risk management plan
  - Cost management plan
  - Schedule management plan
  - Risk register
- Enterprise environmental factors :
  - Organizational process assets
  - And output will be
  - Project documents updates

5.3 Risk Containment

Q. 5.3.1 What is Risk containment ? (Ref. sec. 5.3) **(5 Marks)**

- Containing risk is the process of keeping track of noticed risks, finding new risks, monitoring residual risks and analyzing the risk.
- The inputs for this phase are :
  1. Project management plan
  2. Risk register
  3. Work performance data
  4. Work performance reports
- The output for this phase are :
  1. Work performance information
  2. Change requests
  3. Project management plan updates
  4. Project documents updates
  5. Organizational process assets updates

Syllabus Topic : Risk Projection

5.4 Risk Projection

Q. 5.4.1 Explain process of Risk Projection. (Ref. sec. 5.4) **(5 Marks)**

- There are two ways by which Risk projection (or estimation) attempts to rate each risk :
  - The probability that the risk is real,
  - The consequence of the problems associated with the risk, should it occur.
- The project planner, managers, and technical staff perform four risk projection steps.
- The intent of these steps is to consider risks in a manner that leads to prioritization.
- Be prioritizing risks, the software team can allocate limited resources where they will have the most impact.

Risk Projection/Estimation Steps

1) Establish a scale that reflects the perceived likelihood of a risk (e.g., 1- low, 10-high).
2) Delineate the consequences of the risk.
3) Estimate the impact of the risk on the project and product.
4) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

Contents of a Risk Table

- A risk table provides a project manager with a simple technique for risk projection
- It consists of five columns :
  - **Risk Summary :** Short description of the risk.
  - **Risk Category :** One of seven risk categories.
  - **Probability :** Estimation of risk occurrence based on group input.
  - **Impact :** (1) catastrophic (2) critical (3) marginal (4) negligible.
  - **RMMM :** Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan.

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

Developing a Risk Table

- List all risks in the first column (by the help of the risk item checklists). Mark the category of each risk.
- Estimate the probability of each risk occurring.
- 'Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value.
- Sort the rows by probability and impact in descending order.
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention.

Assessing Risk Impact

Three factors affect the consequences that are likely if a risk does occur :

- **Its nature :** This indicates the problems that are likely if the risk occurs.
- **Its scope :** This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected).
- **Its timing :** This considers when and for how long the impact will be felt.

The overall risk exposure formula is RE = P x C

P = the probability of occurrence for a risk

C = the cost to the project should the risk actually occurs

Example

P = 80% probability that 18 of 60 software components will have to be developed

C = Total cost of developing 18 components is $25,000

RE = 0.80 × $25,000 = $20,000

Syllabus Topic : RMMM

5.5 RMMM

→ (MU - May 15)

Q. 5.5.1	Write note on RMMM. (Ref. sec. 5.5)	May 15, 5 Marks
Q. 5.5.2	Explain the steps involved in setting up or generating RMMM plan. (Ref. sec. 5.5)	(5 Marks)

- 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 mechanisms to assure that there is no time delay in development of work products.
- 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 refocus resources (and manage the project schedule again) to such functions which are completely staffed, enabling newcomers 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.

5.5.1 The RMMM Plan

- It is possible to introduce a risk management strategy in the software project plan, or steps regarding the risk management can be planned into a separate Risk Mitigation, Monitoring, and Management Plan (RMMM).
- The RMMM plan make documentation of all work carried out as a part of risk analysis and referred by the project manager as integral component of the overall project plan.

- Software teams sometimes do not prefer to generate a formal RMMM document. Instead, the risks are documented individually with the help of Risk Information Sheet (RIS).
- Most of the times database system is used to maintain the RIS so that information generation and entry, priority ordering, searches, and remaining analysis may be done easily.
- After documentation of RMMM and the project has begun, risk mitigation as well as monitoring steps starts.
- Risk mitigation is considered as a problem avoidance activity.
- Risk monitoring is considered as a project tracking activity having three primary objectives :
  (1) To assess whether predicted risks do, in fact, occur;
  (2) To make sure that risk aversion steps which are defined for the risk are applied correctly; and
  (3) To gather the data that can be used for future risk analysis.

Syllabus Topic : Software Configuration Management

5.6 Software Configuration Management

Q. 5.6.1 Write note on SCM with benefits and disciplines. (Ref. sec. 5.6) **(10 Marks)**

- Configuration Management (CM) is the process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system lifecycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items.
- Configuration Management is practiced in one form or another as part of any software engineering project where several individuals or organizations have to coordinate their activities.
- While the basic disciplines of Configuration Management are common to both hardware and software engineering projects, there are some differences in emphasis due to the nature of software products.
- Software Configuration Management (SCM) is a system for managing the evolution of software products, both during the initial stages of development and during all stages of maintenance.
- A software product encompasses the complete set of computer programs, procedures, and associated documentation and data designated for delivery to end user.
- All supporting software used in development, even though not part of the software product, should also be controlled by SCM.

5.6.1 Benefits of Software Configuration Management

- SCM provides significant benefits to all projects regardless of size, scope, and complexity.
- Some of the most common benefits experienced by project teams applying the SCM disciplines described here are possible because the SCM system.

- Provides a snapshot of dynamically changing software to help :
  - Make decisions based on an instantaneous view.
  - Determine status at module and system levels.
  - Make better decisions about the future of a software project.
- Tracks concurrent development of modules or components of the overall system to :
  - Prevent different developers from making changes to the same module at the same time,
  - Allow for the overall system progress to be faster,
  - Provide visibility of the entire software project to all developers.
- Organizes all concurrently developing code and associated documentation to :
  - Save overall project time.
  - Focus each phase of the software product development to be organized and executed in a documented, prescribed manner.

5.6.2 SCM Disciplines

- SCM disciplines are used to simultaneously coordinate configurations of many different representations of the software product (e.g., source code, relocatable code, executable code, and documentation).
- In practice, the ways in which SCM disciplines are performed will be different for the various kinds of software being developed (e.g., commercial, embedded, original equipment manufacturer).
- The degree of formal documentation required within and across different lifecycle management phases (e.g., research, product development, operations, and maintenance) will also vary.
- The use of interactive software development environments extends the concepts of SCM to managing evolutionary changes that occur during the routine generation of specifications, designs, and implementation of code, as well as to the more rigidly documented and controlled baselines defined during development and system maintenance.
- The effectiveness of the SCM system increases in proportion to the degree that its disciplines are an explicit part of the normal day-to-day activities of those involved in the software development and maintenance efforts.
- Fig. 5.6.1 presents a diagram showing how the SCM disciplines interact at the highest level.
- The Software Configuration Management Plan (SCM Plan) provides information on the requirements and procedures necessary for the configuration management activities of the software engineering project.
- It specifies who will be responsible for accomplishing the planned activities, what activities will be performed, when the activities will be performed in coordination with the project schedule, and what tools and human resources will be required to perform the activities.
- Consideration should also be given to the overall system configuration.

- The first step in the SCM system is to identify the Configuration Items to be controlled and, as the project progresses, to identify the structure of the intermediate or complete software products.
- Elements to be controlled will normally include the software code and associated documentation.
- Documents could include requirements specifications, designs, user guides, test suites, test results, and user lists.
- Baselines are defined and created in accordance with the SCM Plan and the phase of the project's lifecycle.

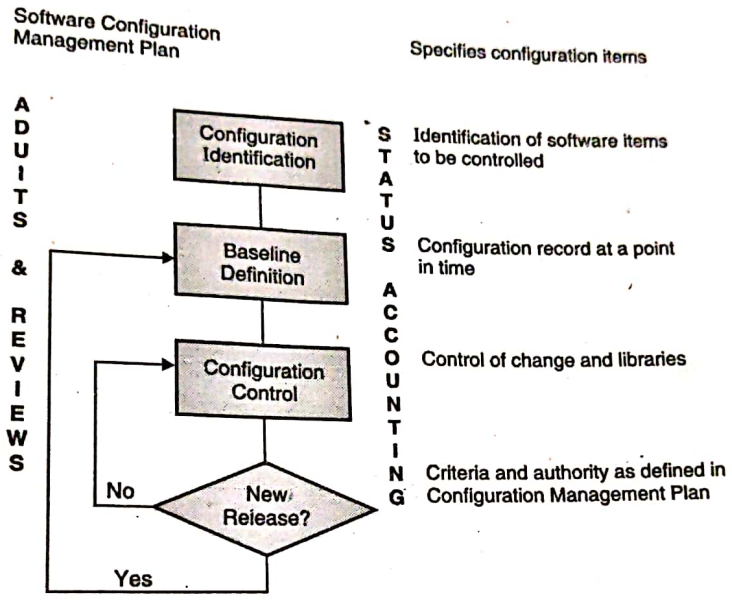


Fig. 5.6.1 : High-Level View of Software Configuration Management

- A baseline is a specification or product that has been formally reviewed and agreed upon, that serves as the basis for further development, and can only be changed through formal change control procedures.
- A baseline is like a snapshot of the aggregate of software components as they exist at a given point in time.
- During each phase of the software lifecycle, configuration items that are completed and accepted by the approval authority become the baseline for that phase.
- Any addition, alteration, or deletion to the approved baseline is believed a change, and is subject to change control.
- Baselines, plus the approved changes from those baselines, constitute the current configuration identification. Each of the baselines established during a software lifecycle controls subsequent software development.
- The following are typical configuration baselines that are established during the life of a software project.

- **Functional Baseline :** Established by the acceptance or approval of the software or system specification. This baseline typically corresponds to the completion of the software requirement review.
- **Allocated Baseline :** Established by approval of the software requirements specification. This baseline typically corresponds to the completion of the software specification review.
- **Developmental Baseline :** Established by the approval of the technical documentation that defines the functional and detailed design (including documentation of interfaces and databases for the computer software). Normally, this baseline corresponds to the time frame spanning the preliminary design review and the critical design review.
- **Product Baseline :** Established by approval of the product specification following completion of the last Formal Functional Configuration Audit (FCA).

- Once configuration items have been identified and baselined, configuration control provides a system to initiate, review, approve, and implement proposed changes.
- Each engineering change or problem report that is initiated against a formally identified configuration item is evaluated to determine its necessity and impact.
- Approved changes are implemented, testing is performed to verify that the change was correctly implemented and that no unexpected side effects have occurred as a result of the change, and the affected documentation is updated and reviewed.
- The discipline of Status Accounting collects information about all configuration management activities.
- The status account should be capable of providing data on the status and history of any configuration item.
- The information maintained in Status Accounting should enable the rebuild of any previous baseline.
- Audits and reviews are conducted to verify that the software product matches the configuration item descriptions in the specifications and documents, and that the package being reviewed is complete.
- Any anomalies found during audits should be corrected and the root cause of the problem identified must be corrected to ensure that the problem does not resurface.
- Generally, there should be a Physical Configuration Audit (PCA) and the Functional Configuration Audit (FCA) of configuration items prior to the release of a software product baseline or an updated version of a product baseline.
- The PCA is performed to determine if all items identified as being part of the configuration are present in the product baseline.
- The audit must also establish that the correct version and revision of each part are included in the product baseline and that they correspond to information contained in the baseline's configuration status report.
- The FCA is performed on each software configuration item to determine that it satisfies the functions defined in the specifications or contracts for which it was developed.

- Appendix C provides a checklist of some of the activities commonly associated with the basic SCM disciplines.
- This list can be modified to include additional activities that are appropriate to the size and complexity of a specific software engineering project.

5.6.3 SCM Repositories

Syllabus Topic : SCM Repositories

Q. 5.6.2 Explain the term "SCM Repositories". (Ref. sec. 5.6.3) **(10 Marks)**

- In the first few years of software engineering, the maintenance of software configuration items was done through paper documents sited in file folders or three-ring binders, and stored in metal cabinets.
- This approach was problematic for many reasons :
  (1) Getting a configuration item whenever required was often difficult,
  (2) Determining the items that were updated, when and by whom was usually tough,
  (3) Building a new version of an already existing application was time wasting and error prone, and
  (4) Narrating thorough or complex relationships among the respective configuration items was virtually not possible.
- Today, SCIs (Software Configuration Items) are maintained in a project database or repository.
- Webster's Dictionary explains the word repository as "anything or person thought of as a center of accumulation or storage".
- In early days of software engineering, the repository was nothing but indeed a person (the developer) who had to memorize the location of entire data relevant to the software project, who had to recall the data that was not noted and reconstruct information that had been lost.
- Although this approach of using an individual as "the center for accumulation and storage" does not act properly.
- Now-a-days, the repository is considered as a "thing" (a database) which works as the center for both accumulation as well as storage regarding software engineering information.
- The role of the individual (the software developer) is to communicate with the repository through tools which are integrated with it.

The Role of the Repository

- The SCM repository is the group of mechanisms and data structures which enable a software team to deal with modifications in an efficient manner.
- It offers the apparent functionality of a modern DBMC (database management system) by the way of taking care of data integrity, sharing, and integration.
- Additionally, the SCM repository provides a hub for the purpose of integration of different available software tools. It helps to manage the flow of the software process, and can implement uniform structure as well as format for software engineering related work products.

- To get these abilities, the repository is described in terms of a meta-model.
- The meta-model is used to identify the way by which information is kept in the repository, the way by which data can be accessed by tools and seen by software developers, how well data security as well as integrity can be maintained, and how simply the present model can be extended to fulfill new requirements.

General Features and Content

- To understand the features and contents of any repository, we have to see it from two perspectives: what are the content which are to be stored in the repository and what are the precise services which are provided by the repository.
- A comprehensive breakdown of sort of representations, documents, and supplementary work products which are kept in the repository is shown in Fig. 5.6.2.

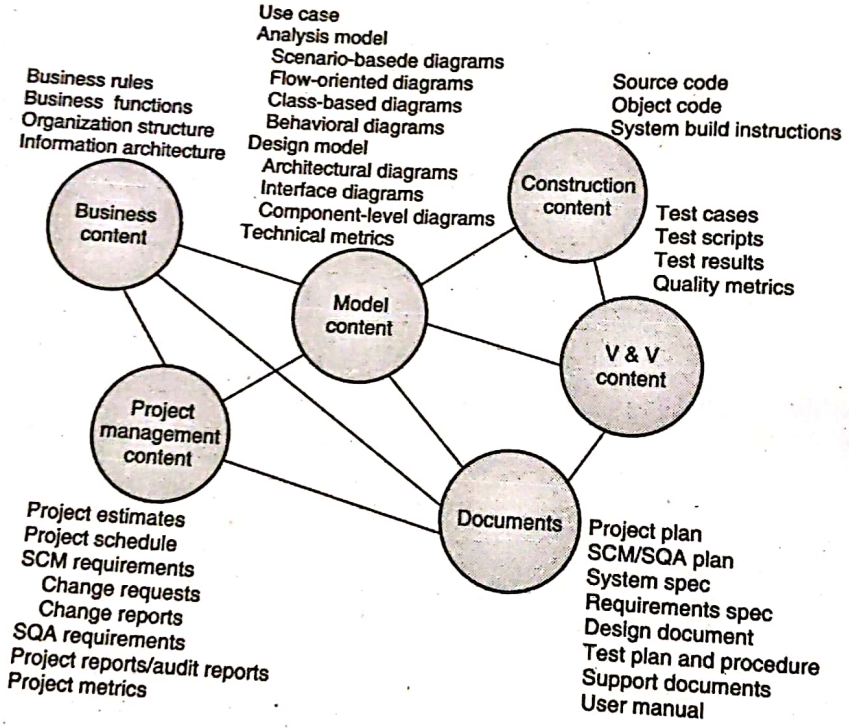


Fig. 5.6.2 : Content of the repository

- There are two different classes of services regarding any robust repository :
  (1) The similar types of services which may be required by any sophisticated DBMS and
  (2) The services which are specific to the environment of software engineering.
- A repository that serves a software engineering team should also :
  (1) Incorporate with or directly support process management functions,
  (2) Support particular rules which govern the SCM function and the information which is maintained inside the repository,
  (3) Provide an interface to other tools which are related to software engineering, and

(4) Maintain storage of several types of sophisticated data objects such as text, graphics, video, audio, etc.

Syllabus Topic : SCM Process

5.6.4 SCM Process

- The SCM process defines a sequence of tasks which have 4 most important objectives :
  (1) To identify all of the respective items which jointly define the software configuration,
  (2) To manage changes applied to these items,
  (3) To ease the building of different versions of an application, and
  (4) To make it sure that there is consistency in software quality as the configuration evolves over time.
- A process which is able to achieve these objectives need not be technical or heavy, but it must be characterized in such a way that it should enable a software team to solve all of the complex questions :
  - What will be the way for the software team to recognize the distinct elements of a software configuration?
  - What will be the way for the organization to manage the several existing versions of an application that will help to accommodate change efficiently?
  - What will be the way for the organization to control changes prior and later the software is released to a customer?
  - Who are individuals responsible for approving as well as ranking requested changes?
  - How can we conform that the respective changes have been made correctly?
  - What will be the mechanism to apprise others of changes that are made?
- These questions lead to the definition of five SCM tasks :
  1. Identification,
  2. Version control,
  3. Change control,
  4. Configuration auditing,
  5. Reporting

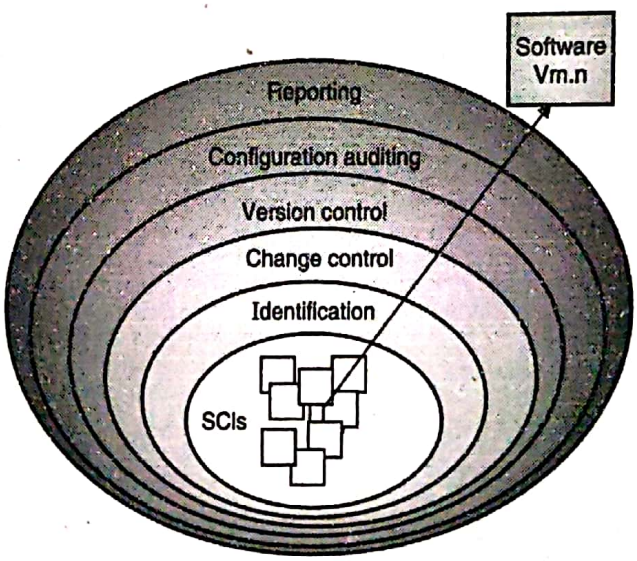


Fig. 5.6.3 : SCM Tasks

5.6.4(A) Configuration Identification

Q. 5.6.3 What is Configuration Identification in SCM ? (Ref. sec. 5.6.4(A)) **(5 Marks)**

- The first step in the SCM system is the identification of the items to be managed and the specification of the management authority responsible for each item.
- This is a critical step in the system since the identification scheme is employed throughout the lifecycle of the software product.
- The levels of authority assigned responsibility for each configuration item will vary depending on the complexity of the software product, the size of the development team, and the management style of the responsible organization.
- Configuration identification consists of selecting the configuration items and recording their functional and physical characteristics as set forth in technical documentation such as specifications, drawings, and associated lists.
- The following are examples of items managed in the SCM system.
  - Management plans
  - Specifications (requirements, design)
  - User documentation
  - Test plans, test design, case, and procedure specifications
  - Test data and test generation procedures
  - Support software used in development, even though not part of the delivered software product
  - Data dictionaries and various cross-references
  - Source code (on machine-readable media)
  - Executable code (the run-time system)
  - Libraries
  - Databases (data that are processed and data that are part of a program)
  - Maintenance documentation (e.g., listings, detail design descriptions)
- Effective management of the development of a software product requires careful definition of the configuration items.
- Changes to these items also need to be defined since these changes, along with the project baselines specify the evolution of the software product.
- SCM includes all of the entities of the software product as well as their various representations in documentation.
- The entities are not all subject to the same SCM disciplines at the same time.
- Support software is the class of software that may or may not be delivered with the software product, but is necessary for designing, enhancing, or testing the changes made during the life of the product.

- Support software may be user-furnished, developed in-house, leased from a vendor, or purchased off-the-shelf.
- In SCM, the focus is on describing the necessary controls used to manage support software.
- Developers and maintainers need to ensure that the support software is available for use for as long as the software product is in use.
- Whenever a production baseline is established, it is very important to archive all environmental and support tools along with the production code.
- Software tools used in development, especially compilers and middleware, should be placed under configuration control so their identities and versions are included in the baseline data about each release.
- Consideration also needs to be given to the hierarchy of entities managed during the SCM system. There are several different ways of structuring this hierarchy.
- One way is a three-level hierarchy consisting of configuration items, components, and units.
- Another way of structuring the entities is in terms of the interrelationships between the software being developed and the other software entities used during development and testing such as support software, test software, and the operating system.
- A third method for structuring entities is in terms of the intermediate products generated in the process of building the software product, such as: modifiable entities (e.g., source code, document, test data); compilation entities (e.g., compilers, support software); and configuration items (e.g., different representations created in the process of producing deliverables).
- An important aspect of configuration identification is the use of a formal naming convention for each entity.
- This naming convention, which typically uses a combination of mnemonic labels and version numbers, should be applied to all components of a configuration item.
- In establishing the naming convention, consideration should be given to system constraints such as name length or composition.
- Consistency in the application of this identification scheme is critical to accurate tracking of the software engineering process.

5.6.4(B) Change Control (Configuration Control)

→ **(MU - Dec. 15, May 17, May 18)**

Q. 5.6.4	Explain change control activities in SCM. (Ref. sec. 5.6.4(B))	Dec. 15, May 17, 5 Marks
Q. 5.6.5	Explain steps in change control. (Ref. sec. 5.6.4(B))	May 18, 5 Marks

- 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 :
  - Defining the information needed for approving the requested change.
  - Identifying the review process and routing of information.
  - Describing the control of the libraries used to process the change.
  - Developing a procedure for implementing each change in the code, the documentation, and the released software product.

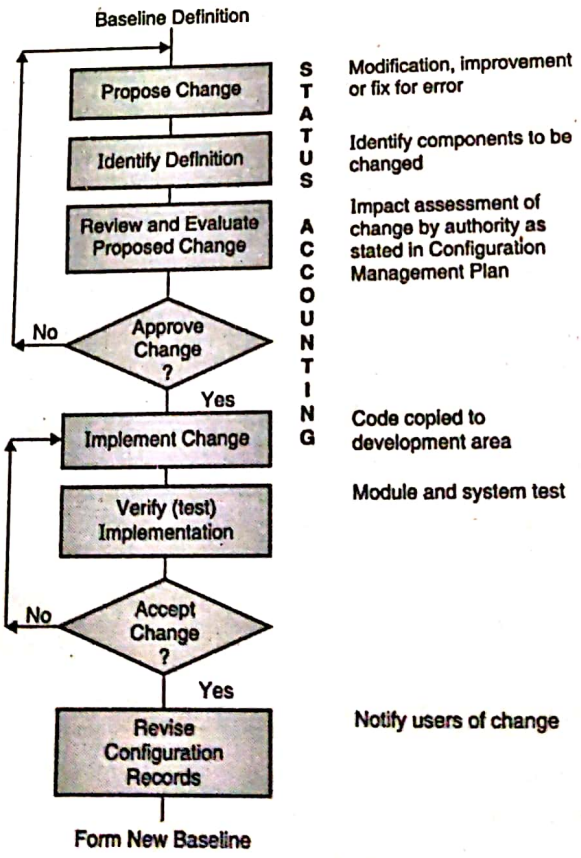


Fig. 5.6.4 : Configuration Control Process

5.6.4(C) Version Control

Q. 5.6.6 Explain version control activities in SCM. (Ref. sec. 5.6.4(C)) → (MU - Dec. 15, May 17, May 18)

Q. 5.6.7 Explain steps in version control. (Ref. sec. 5.6.4(C)) Dec. 15, May 17, 5 Marks. May 18, 5 Marks

- 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 :
  - A project database (repository) which holds all the appropriate configuration objects,
  - A version management capability which holds all the relevant versions of a configuration object,
  - 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.

5.6.4(D) Configuration Audit

Q. 5.6.8 What is Configuration Audit in SCM ? (Ref. sec. 5.6.4(D)) (5 Marks)

- Identification, version control, and change control are the processes which help user to maintain sequence in what would otherwise be a confused situation.
- However change is tracked by even most successful control mechanisms only until generation of ECO (Engineering Change Order).

- What will be the way by which a software team make it sure that the change has been appropriately implemented?
- The answer has two aspects :

  (1) Technical reviews and

  (2) The software configuration audit.
- The focus of technical review is on the technical accuracy of the configuration object in which modifications are done.
- The reviewers assess the SCI (Software Configuration Items) for the purpose of determining consistency with other SCIs, omissions, or potential side effects.
- There is need to conduct a technical review for all but the most trivial changes.
- A software configuration audit in general complements the technical review by the way of assessing a configuration object for characteristics which are usually ignored in the process of review.
- The audit asks and answers the following questions :

  1. Has the modifications mentioned in the ECO been made? Have any additional modifications been incorporated?
  2. Is there any process of conducting technical review for assessing technical correctness?
  3. Has the software process been followed and is the application of software engineering standards done properly?
  4. Has the change been focused in the SCI? Have there is specification of change date and change author? Do the change is reflected by attributes of the configuration object?
  5. Has there is implementation of SCM procedures for noting the change, recording it, and reporting?
  6. Has there is proper updations in all related SCIs?
- Sometimes, the audit questions are asked as an integral part in the process of technical review.
- However, when SCM is a formal activity, the quality assurance group conducts the configuration audit separately.
- This formal configuration audits also make it sure that the accurate SCIs have been integrated into a precise build and that the whole documentation is up-to-date and consistent with the currently built version.

5.6.4(E) Status Reporting

- Configuration Status Reporting (CSR) which is also known as status accounting is an SCM task that gives answers of following questions :

  1. What happened?
  2. Who did it?
  3. When did it happen?

4. What else will be affected?

- A CSR entry is made whenever a new or updated identification is assigned to SCI
- A CSR entry is made whenever the CCA (Change Control Authority) approve a change.
- Whenever a configuration audit is carried out, the results are reported as part of the CSR task.
- It is possible to keep the output from CSR in an online database or website, because of which it will be easy for software developers or support staff to access the change information by keyword category.
- Additionally, generation of a CSR report is done regularly and is intended to give latest information regarding important changes to management as well as the practitioners.

5.7 Software Quality Management

Q. 5.7.1 Write note on Software Quality Management. (Ref. sec. 5.7) **(10 Marks)**

- **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.

5.7.1 Activities of Software Quality Management

Fig. 5.7.1 : 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 :

5.7.2 Software Functional Quality

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

5.7.3 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.

5.7.4 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.

5.7.5 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 :

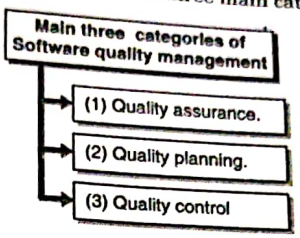


Fig. 5.7.2 : Main categories of Software Quality Management

➔ **(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.

5.7.6 Software Quality Challenge

Q. 5.7.2 What are the Challenges in Software Quality Management ? (Ref. sec. 5.7.6) **(5 Marks)**

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.
- Software packages allow millions of operational possibilities. Therefore, assuring the operational potential correctly is a major task to the software industry.

(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.

5.7.7 Process Quality Product

Q. 5.7.3 Explain the concept of Quality Products in details. (Ref. sec. 5.7.7) **(5 Marks)**

- 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. 5.7.3 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.

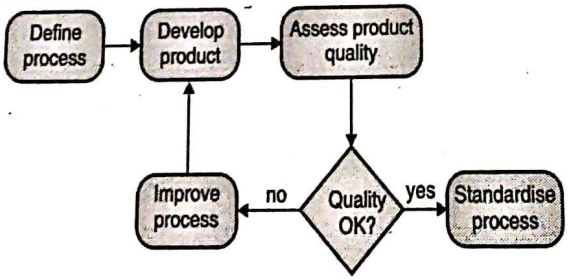


Fig. 5.7.3 : Process based quality assessment

5.8 Software Quality Assurance

Q. 5.8.1 Explain the concept of Software Quality Assurance in details. (Ref. sec. 5.8) **(5 Marks)**

- Software Quality Assurance (SQA) is a group of activities to guarantee that the quality in software engineering processes is maintained.
- It make sure that developed software project would meet and complies with the predefined or standardized quality specifications.
- SQA is an ongoing activity in the Software Development Life Cycle (SDLC) that consistently checks the developed software to guarantee that it should meet the desired quality standards.

- SQA practices are adapted in most types of software development industries.
- SQA incorporates and implements software testing methodologies to test the software. Rather than every time checking for quality after completion of each project, SQA processes test the quality in each phase of development, throughout the software development process.
- With SQA, the software development activities moves into the next level only if the present/prior level complies with the required quality standards.
- SQA generally works on one or more engineering standards that assist in building software quality guiding principle and execution strategies.
- Once the processes have been defined and implemented, Quality Assurance has the following tasks :
  (a) To identify the weaknesses in the processes,
  (b) Correct those weaknesses to constantly improve the process.

It includes the following activities

(1) Process definition and accomplishment
(2) Auditing
(3) Guidance

Processes could be

(1) Software Development method
(2) Project Management
(3) Configuration Management
(4) Requirements Development/Management
(5) Estimation
(6) Software Design
(7) Testing, etc.

5.8.1 Components of SQA System

SQA system always combines a broad variety of SQA components. These components can be classified into the following six classes.

(1) Pre-project components

- This guarantees that the project commitments have been clearly defined taking into account the required resources, the schedule and budget.
- The improvement and quality plans have been correctly determined.

(2) Components of project life cycle activities assessment

- The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage.

- The development life cycle stage components identify the design and programming flaws. Its components are divided into the following sub-classes : Reviews, Expert opinions, and Software testing.
- The SQA components used during the operation–maintenance phase include specialized maintenance components as well as development life cycle components, which are applied mainly for functionality to improve the maintenance tasks.

(3) Components of infrastructure defect avoidance and improvement

The main aim of these components is to remove or at least minimize the rate of errors, based on the organization's SQA experience.

(4) Components of software quality management

This class of components deals with a number of goals, such as the managing the development and maintenance of activities and early managerial support actions that mainly avoid or decrease schedule and budget failures and their consequences.

(5) Components of standardization, certification, and SQA system evaluation

- These components adapt international professional and managerial values within the institute.
- The main motive of this class is to use international professional understanding, enhancement of the organizational quality systems with other industry, and evaluation of the achievements of quality systems according to a general scale.
- The different standards can be categorized into two main groups: quality management standards and project process standards.

(6) Organizing for SQA - the human components

- The SQA base includes managers, testing professionals, the SQA unit and the resources involved in software quality such as SQA trustees, SQA committee members, and SQA meeting members.
- Their main aims are to make the first move and maintain the implementation of SQA components, defect deviations from SQA procedures and methodology, and advise some improvements.

5.8.2 Pre-project Software Quality Assurance

Q. 5.8.2 What is concept behind the Pre-Project quality assurance ? (Ref. sec. 5.8.2) **(5 Marks)**

These components help to improve the groundwork task taken before starting a project. It includes :

(1) Deal Review

(2) Development and Quality Plans

(1) Deal Review

- Software is developed for an agreement negotiated with a client or for an internal order to build up a firmware to be fixed within a hardware product.

- In all these cases, the development division is loyal to an approved functional specification, budget and schedule. Hence, contract/deal review actions must include in depth examination of the project tender draft and the contract drafts.
- Specifically, contract review activities include :
  - Clarification of the clients requirements,
  - Review of the project's schedule and resource requirement estimates,
  - Assessment of the professional resources ability to carry out the planned project,
  - Evaluation of the customer's capacity to fulfill his obligations,
  - Evaluation of development risks.

(2) Development and Quality Plans

- After signing the software development contract with an industry or an internal department of the same organization, a development plan of the project and its integrated quality assurance activities are prepared.
- These plans consist of extra particulars and required revisions based on previous plans that have provided the base for the current proposal and contract.
- Most of the time, it takes many months between the contract submission and the signing of the contract. During this phase, resources such as staff availability and professional capabilities may get changed.
- The plans are then revised to reflect the changes that occurred in the interim.

(i) The main problems treated in the project development plan

(1) Schedules

(2) Important manpower and hardware resources

(3) Risk evaluations

(4) Organizational problems : team members, subcontractors and partnerships

(5) Project methodology, development tools, etc.

(6) Software reuse plans.

(ii) The main issues treated in the project's quality plan

(1) Quality aims

(2) Criteria for initial and final project stage

(3) Lists of reviews, tests, and other scheduled confirmation and validation activities.

Syllabus Topic : Metrics

5.8.3 Metrics

→ (MU - May 15)

Q. 5.8.3 Explain different metrics used for maintaining software quality. (Ref. sec. 5.8.3) May 15, 10 Marks

Software metrics can be classified into three categories

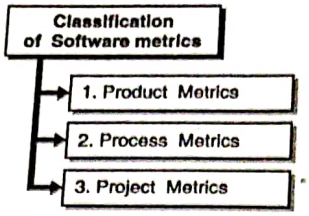


Fig. 5.8.1 : Classification of software metrics

➜ **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 :

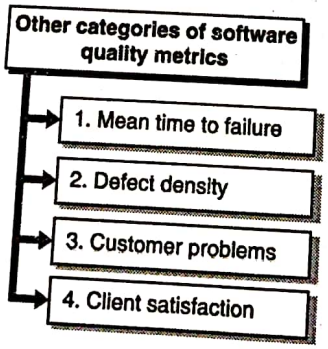


Fig. 5.8.2 : Other categories of software quality metrics

➜ **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.

5.8.4 In-process Quality Metrics

In-process quality metrics is the metric which track the fault during the arrival of machine testing for organizations. This metric includes :

- Defect intensity while doing machine testing.
- Defect arrival method during machine testing.
- Phase-based defect removal pattern.
- Defect elimination efficiency.

5.8.5 Defect Density During Machine Testing

- Defect velocity during machine testing (testing after code is integrated into the system) is associated with the defect velocity in the field.

- If the higher defect velocity is established during testing then it is a notification that the software has practiced higher error injection during its development process.
- It is generally useful to monitor succeeding releases of a product in the similar development organization.

5.8.6 Defect Arrival Pattern During Testing

Q. 5.8.4 What is Defect Arrival Pattern ? (Ref. sec. 5.8.6) **(2 Marks)**

Defect arrival pattern during machine testing

- The overall defect rate during testing will give the summary of the defects.
- The different methods of defect arrivals in the project will give additional information about different quality levels in the project.
- It includes the following :
  1. The defect detection or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be exact defects.
  2. The pattern of applicable defect arrivals when problem identification is done on the reported problems. This is the true defect.
  3. The pattern of defect backlog overtime. This pattern is required because development organizations cannot find and fix all the identified problems instantly.
  4. If the defect backlog rate is large at the ending of the development series in the project and a lot of fixes have to be incorporated into the system process, the consistency of the system (hence its quality) will be affected.

5.8.7 Phase-Based Defect Removal Pattern

- This is an expansion of the defect density while doing testing.
- In addition to testing, it measures the defects at all stages of the development cycle in the project, counts the design reviews, code reviews, and do verifications before testing.
- Because a large percentage of development defects is associated to design problems, conducting formal reviews, or functional verifications to improve the defect elimination ability of the process at the front-end minimizes errors in the software.
- The phase-based defect elimination reflects the overall defect elimination skill of the development process.
- With consideration to the pattern for the design and coding phases, in addition to defect density, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

5.8.8 Maintenance Quality Metrics

Following are the fixes that can be carried out to remove the defects as soon as possible with outstanding fix quality.

- Fix backlog and backlog management directory,
- Fix response time,
- Percent offending fixes,
- Fix quality.

5.8.9 Fix Backlog and Backlog Management Index

- Fix backlog is associated to the velocity of bug arrivals and the rate at which fixes are done for identified problems.
- It is a simple tally of identified problems that stay behind at the end of each month or each week.
- When same is used in the format of a chart, the chart can provide significant information for organizing the continuation process.
- Backlog Management Index (BMI) is used to manage the backlog of current and unresolved bugs.
- If BMI is larger than 100, it means the backlog is minimized. If BMI is less than 100, then the backlog is improved.

5.8.10 Fix Quality

- Fix quality or the number of faulty fixes is another important quality metric for the maintenance phase in the SQA.
- A fix is flawed if it did not fix the identified problem, or if solved the original problem but created a new bug. For complex software, defective fixes are harmful to client satisfaction. The metric of percent flawed fixes is the percentage of all fixes in a time period that is flawed.

5.8.11 Defective Fix

- Trace it in the month it was found or trace it in the month the fix was given. The first is a customer measure; the second is a process measure.
- The distinction between the two dates is the period of the defective fix.
- QA helps to make sure that the development of high-quality software is in progress.
- SQA practices are implemented in most types of software development, in spite of which fundamental software development model being used. In a broader category, SQA includes and implements software testing technologies to test the software.
- Rather than checking for quality after conclusion, SQA processes test for quality in each phase of development until the software is complete. With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards.
- SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies. These standards include the ISO 9000 and Capability Maturity Model Integration (CMMI).

5.9 SQA Processes

Q. 5.9.1 Explain SQL Process with its benefits. (Ref. sec. 5.9) **(5 Marks)**

- Processes include :
  1. Software Development Methodology
  2. Project Management
  3. Configuration Management
  4. Requirements Development/Management
  5. Estimation
  6. Software Design
  7. Testing
- Once the processes have been defined and implemented, Quality Assurance has the following responsibilities :
  (1) Identifying weaknesses in the processes,
  (2) Correcting those weaknesses to continually improve the processes.
- The quality management system under which the software system is created is normally based on one or more of the following models/standards :
  (1) CMMI
  (2) Six Sigma
  (3) ISO 9000

5.10 Benefits of SQA

- Monitoring and improving process.
- Make sure that he standards and procedure are followed.
- Prevents the significant problems from occurring.

Syllabus Topic : Software Quality Assurance Task and Plan

5.11 Software Quality Assurance Task and Plan

Q. 5.11.1 List Various Activity in SQA. (Ref. sec. 5.11) **(4 Marks)**

Q. 5.11.2 What are the different quality assurance guidelines and describe them ? (Ref. sec. 5.11) **(4 Marks)**

5.11.1 Test Management Reviews and Audit

Management Review

- Management Review is also known as Software Quality Assurance or (SQA).

- It mainly focuses on the software method rather than the software related work products.
- Quality Assurance is a set of activities planned to guarantee that the project manager follows the regular process which is already defined in the system.
- In other words, Quality Assurance makes sure the Test Manager is doing the right things in the right way.
- **Audit :** An audit is the assessment of the work products and associated data to assess whether the regular process was carried out or not.

5.11.2 Implementation of the Quality Assurance

Q. 5.11.3 Explain the steps involved in implementing the quality assurance. (Ref. sec. 5.11.2) **(10 Marks)**

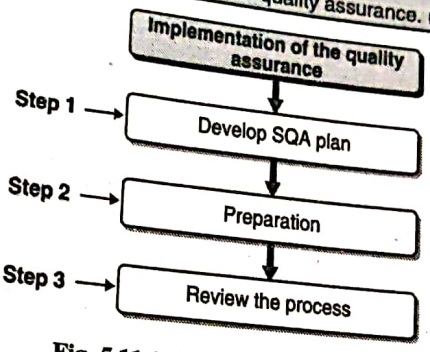


Fig. 5.11.1 : Step for SQA Plan

1. Develop SQA Plan

- As testing processes need Test Plan, similarly the SQA processes need **a plan** which is called **SQA plan.**
- The main motive of SQA plan is to plan the processes and measures to make sure that products are developed.
- During project planning, Test Manager makes an SQA plan where SQA audit is scheduled periodically.
- In the SQA Plan, the Test Manager should do following :

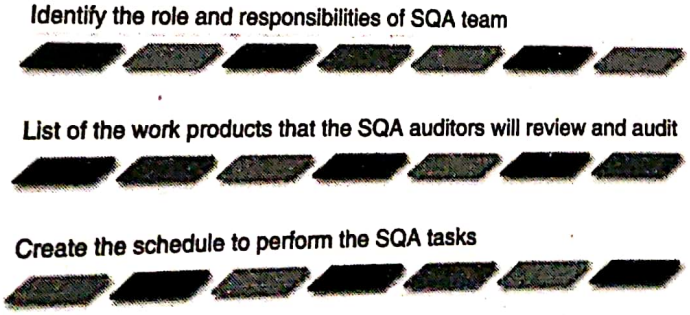


Fig. 5.11.2 : Role and Responsibility of SQA

1. **Identify the role and responsibilities of SQA team**
   - In a project team, every member must have responsibility for the quality of his or her work. Each person has to make sure their work meets the QA criteria.
   - The SQA team is the group of persons who plays the **major role in** the project. Without QA, no business will run successfully. Therefore, the Test Manager has to make clear the **responsibility** of each SQA member in SQA plan as below :
     - **Review** and **evaluate** the quality of project activities to meet the QA criteria
     - **Coordinate** with management board and project teams to assess requirements and engage in project review and status meetings.
     - **Design track** and **collect** metrics to monitor project quality.
     - **Measure** the quality of product; **ensure** the product meets the customer expectations.
2. **List of the work products that the SQA auditor will review and audit**
   - The Test Manager should list out all the work products of each Test Management Process.
   - Define which facilities or equipment the SQA auditor can access to perform SQA tasks such as process evaluations and audits.
3. **Create the schedule to perform the SQA tasks**
   - In this step, the Test Manager should describe the tasks to be performed by SQA with special importance on SQA activities as well as the work for each task.
   - Test Manager also creates the scheduling of those SQA tasks. Normally, the SQA schedule is driven by the project development schedule.
   - Therefore, a SQA task is performed in relation to which software development activities are taking place.

☞ **Define the standards/methodology**

To review the Management activities against the standards process, following steps are taken :

- Define the policies and procedures intended to prevent defects from the management process,
- Document the policies and procedures,
- Inform and train the staff to use these processes.

5.11.3 Review the Process

Review project activities are to verify compliance with the defined management process.

In the management review, the SQA members have to perform 5 SQA reviews as follows :

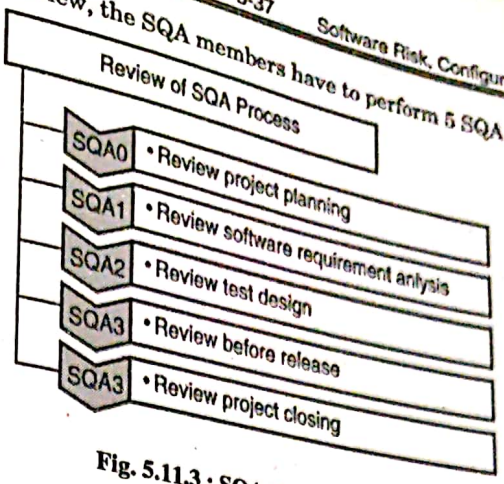


Fig. 5.11.3 : SQA Review Process

5.12 Quality Evaluation Standards

Q. 5.12.1 What are the Quality Evaluation Standards ? (Ref. sec. 5.12) (2 Marks)

5.12.1 Quality Evaluation Standards

- In the recent years, Software quality is getting great importance, mainly because of the very vital role played by software in our day-to-day lives.
- It is necessary to conduct evaluations of the software products. The AQC Lab was established for this purpose and its core responsibility is evaluating the quality of software products against the ISO/IEC 25000 standard.
- Although numerous certifications for software quality exist (e.g., ISO/IEC15504, CMMI, etc.), there is little evidence to suggest that compliance with any of these standards guarantees good software products.
- Critics have gone so far as to suggest that the only thing these standards guarantee is uniformity of output and thus, may actually lead to the production of bad products.
- Consequently, the idea that software evaluations should be based on direct evidence of product attributes is becoming more widespread. Therefore, a growing number of organizations are becoming concerned about the quality of the products that they develop and/or acquire, as well as the processes.

5.13 Six Sigma

Q. 5.13.1 What is the philosophy of six sigma ? Explain six sigma strategies. (Ref. sec. 5.13) (8 Marks)

Q. 5.13.2 Explain six sigma for software engineering. (Ref. sec. 5.13) (4 Marks)

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

5.13.1 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 idea behind Six Sigma is if anyone can evaluate how many "defects" can have in a process, it can be thoroughly figure out how to minimize them and can go closer to "zero defects" as possible and particularly that means 99.9997% perfect.

5.13.2 Key Concepts of Six Sigma

Six Sigma revolves has a few key concepts.

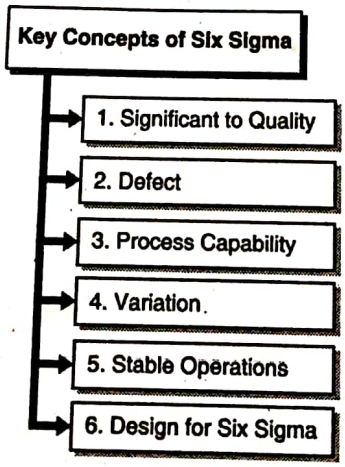


Fig. 5.13.1 : Key concepts of six sigma

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.

5.13.3 Benefits of Six Sigma

Following are the benefits of Six Sigma :

- Generates persistent success,
- Sets a goal for everyone,
- Improves value to customers,
- Improves the rate of improvement,
- Promotes learning.

5.13.4 Elements of Six Sigma

There are three main elements of Six Sigma :

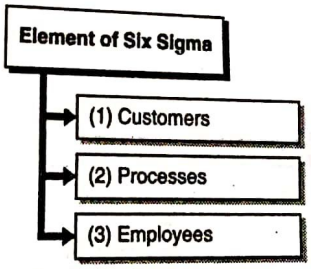


Fig. 5.13.2 : Elements of six sigma

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

- An industry must involve all its employees in the Six Sigma process.
- Company can provide opportunities and incentives for employees to concentrate on their talents and ability to satisfy customers and ultimately the company.
- The important factor in the Six Sigma is that all the team members have proper roles and correct objectives.

5.13.5 Responsibilities/roles in a Six Sigma

There are some more responsibilities/roles in a Six Sigma program, which are as follows :

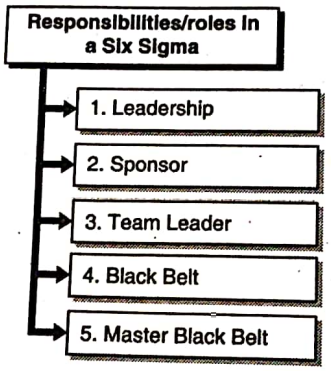


Fig. 5.13.3 : Responsibilities/roles in a Six Sigma

1. Leadership

A leadership team always defines the goals and objectives in 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.

Syllabus Topic : Software Reliability

5.14 Software Reliability

Q. 5.14.1 Write note on Software Reliability. (Ref. sec. 5.14) **(5 Marks)**

- In overall quality of software, reliability is considered as an important aspect.
- If there is frequent failure in performance of a program, it affects the quality of software.
- Unlike other quality factors, it is possible to measure the Software reliability directly and estimated with the help of historical and developmental data.
- Statistically Software reliability is defined as "the probability of failure-free operation of a computer program in a specified environment for a specified time".
- Now to better understand consider a program X is estimated to have a reliability of 0.999 over 8 elapsed processing hours.
- That means, if there is need to execute program X 1000 times and needs a total of 8 hours of elapsed processing time (execution time), it is possibly execute correctly (without any of the failure) 999 times.
- In software reliability one important question is: What is meant by the term failure?
- In the context of any type of discussion regarding software quality and reliability, failure is considered as non-fulfillment of software requirements.
- However, even within this definition, there are several aspects. Failures can be only frustrating or terrible.
- It is possible to correct some failures within seconds, whereas some other may need weeks or even months to correct.
- Further if the issue is complicated, correction made in one failure may leads to generation of other errors.

5.14.1 Measures of Reliability and Availability

- Early work in the aspect of software reliability tried to extrapolate the mathematics regarding hardware reliability theory for the calculation of software reliability.

- The prediction of most hardware-related reliability models is done on failure because of wear instead of failure caused by design defects.
- In case of hardware, possibility of failures because of physical wear such as the impact of temperature, corrosion, shock are more likely as compared to design-related failure.
- Unluckily, the reverse is true for software. Indeed, all of the failures regarding software can be traced to design or implementation problems; since there is no any scope of wear.
- In case of a computer-based system, MTBF (meantime-between-failure) is a simple measure of reliability.

  TBF = MTTF + MTTR

  where

  MTTF = mean-time-to-failure and

  MTTR = mean-time-to-repair
- Number of researchers consider that MTBF is a extreme useful measure in comparison with other quality-related software metrics.
- Straightforwardly, end users are practically concerned with failures; they do not have to do anything with defect count.
- Since each defect which has been contained within a program do not has similar failure rate, the sum of all the defect count gives very little indication regarding the reliability of a system.
- We will consider a computer-based program that has been in operation for three thousand processor hours without any kind of failure.
- Several defects in this computer-based program may not be detected for tens of thousands of hours before their discovery.
- In such case the MTBF of those errors might be thirty thousand or even sixty thousand processor hours.
- Remaining defects, which are to be yet undiscovered, might have a failure rate of four thousand or five thousand hours.
- Even though all of the first category of errors (having long MTBF) are removed, the effect on software reliability is minor.
- But, MTBF can be considered as problematic for two reasons :

  (1) It estimates a time span in between failures, but not able to provide projected failure rate, and

  (2) MTBF can be misinterpreted to mean average life span even if that is not what it implies.
- Another measure of reliability is FIT (failures-in-time). It is a statistical measure of number of failures a component will have over one billion hours of operation.
- Therefore, 1 FIT is considered as equivalent to single failure in the span of every billion hours of operation.
- Besides a reliability measure, one should also develop a measure of availability.

- Software availability is nothing but the probability which a program is operating as per the requirements at a given point in time and is defined as :

  Availability = (MTTF / (MTTF+MTTR)) * 100%

- The sensitivity of MTBF reliability measure is similar to MTTF and MTTR.
- The availability measure is considered as little bit more sensitive compared to MTTR, an indirect measure of the maintainability of software.

Syllabus Topic : Formal Technical Review (FTR)

5.15 Formal Technical Review (FTR)

Q. 5.15.1 What is FTR in SQA? What are its objectives? Explain steps in FTR. (Ref. sec. 5.15) → (MU - Dec. 15, May 16, May 17, May 18)

Q. 5.15.2 Explain FTR. (Ref. sec. 5.15) Dec. 15, May 17, 10 Marks
May 16, May 18, 3 Marks

- 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.

5.15.1 Steps in FTR

→ (MU - May 18)

Q. 5.15.3 Explain the review guidelines considered during FTR. (Ref. sec. 5.15.1) May 18, 8 Marks

1. The review meeting

- Every review meeting should be conducted by considering the following constraints :
  - **Involvement of people :** Between 3 to 5 people should be involve in the review.

- **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
- **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.
- You should 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.

5.16 Walkthrough

Syllabus Topic : Walkthrough

Q. 5.16.1 Explain Walkthrough. (Ref. sec. 5.16) → (MU - May 16) May 16, 3 Marks

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

5.17 Comparison between FTR and Walkthrough

→ (MU - May 16)

Q. 5.17.1 Compare FTR and Walkthrough. (Ref. sec. 5.17) May 16, 4 Marks

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

Parameter	FTR	Walkthrough
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 Ends...

❑❑❑

CHAPTER 6

Module 6

Software Testing and Maintenance

Syllabus

Strategic Approach to Software Testing, Unit testing, Integration testing, Verification, Validation Testing, System Testing, Software Testing Fundamentals, White-Box Testing, Basis Path Testing, Control Structure Testing Black Box Testing, Software maintenance and its types, Software Re-engineering, Reverse Engineering.

6.1 Software Testing

- Software testing is a procedure to verify whether the actual results are same as of expected results.
- Software testing is performed to provide assurance that the software system does not contain any defects.
- Defects means errors which are detected by tester when actual result of our software module is not identical as expected result.
- Software testing helps us to find out whether all user requirements are fulfilled by our software or not.
- Software quality depends on; at what extend software fulfills user requirements and number of defects occurred in software.
- Software testing is used to give assurance that we deliver quality product to customer.
- To perform software testing we create test cases and test data. Test Case is a collection of actions which applied on our software product to check specific feature or functionality of it.
- Collection of test cases is called as test suit.
- Test data is the input provided to modules which are present in our software product. Test data represents the data which effects execution of the particular module. Sometime test data is used for positive testing. That means it is used to check that a provided set of input for given function generate expected result or not. Sometime test data is used for negative testing. That means test data is used to test the capability of our software modules to handle unexpected input.

6.1.1 Advantages of Software Testing

Q. 6.1.1 What are the advantages of Software Testing ? (Ref. sec. 6.1.1) **(5 Marks)**

There are several advantages of software testing :

1. Testing reduces the possibility of software failure by removing errors which leads to software failure.
2. Testing process removes maximum possible errors from software and helps to deliver qualitative software to customer.
3. Testing ensures correctness and completeness of software along with quality.
4. Testing allows us to verify and validate the software and ensures that software will satisfy all of the user's requirements.
5. Testing enables the software to produce expected outcome by removing errors which prevent system from producing expected outcome.

6.1.2 Types of Software Testing

Q. 6.1.2 Write note on Manual and Automation Testing. (Ref. sec. 6.1.2) **(10 Marks)**

Software Testing is divided in to two categories :

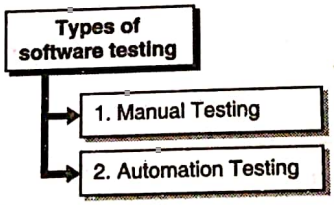


Fig. 6.1.1 : Types of software testing

➜ 1. Manual Testing

- Manual Testing is one of the type of Software Testing in which Tester treat himself as end user and use every module of application from login module to log out module to check application behavior according to requirement provided by user.
- In manual testing, tester manually runs all the test cases without help of any automation testing tools.
- Manual Testing is the very basic type of all testing types and supports to search the bug (error) in the software system.
- First we do manual testing for any new application before going for automation testing. We perform feasibility study of automation testing in manual testing i.e. do the analysis of benefits of performing automation testing for our project to take the decision that whether to do automation testing or not.
- For performing manual testing, we do not need knowledge of any testing tool.
- Manual Testing requires more effort, but is necessary to check automation feasibility.

- One important thing to know about Software Testing is that **"100% Automation is not possible"** so we need to perform manual testing.

Goals of Manual Testing

- The main purpose of performing manual testing is to give assurance that our software product does not contain any defect and it fulfills all functional requirements of end user.
- Test Suites (i.e. collection of test cases) or cases, are designed in the testing phase and they must test all functionalities present in our software product.
- Manual testing gives assurance that reported defects are fixed by developer and tester performs retesting on it after fixing the defects.
- Mainly manual testing verifies the quality of the software product and deploys bug-free software to the customer.
- List of manual testing is as follow :
  - (i) Unit testing
  - (ii) Integration testing
  - (iii) System testing
  - (iv) Acceptance testing

2. Automation Testing

- Automation Testing is a process in which we use automation tools to run our test cases to find out bugs from our software product.
- The automation software is able to enter test data into the software on which we perform testing and it also matches the expected and actual results to create test reports.
- Automation testing requires considerable amount of money and resources such as employees, testing tools etc.
- After change in requirement or occurrence of error, we need to make changes in code and we require executing same test suite again and again.
- We can record test suite by using test automation tools and re-play it when needed.
- Automation testing does not require human interaction to perform testing like manual testing.
- Aim of Automation testing is to decrease number of test cases to be executed manually but not replace Manual Testing completely.
- To perform automation testing we require generation of test cases, test data and test script.
- Automation testing is performed for project if requirements of projects are stable at some extent i.e. requirements are not frequently changing.
- List of some automation tools
  - (i) Selenium
  - (ii) Mentis
  - (iii) Quality Test Professional (QTP)
  - (iv) Buxila
  - (v) HP ALM (Application Lifecycle Management)

6.1.3 Difference between Manual and Automation Testing

Q. 6.1.3 Differentiate between manual and automation testing. (Ref. sec. 6.1.3) **(5 Marks)**

Parameter	Manual Testing	Automation Testing
Human interaction	To perform manual testing human interaction is required for test execution.	To perform Automation Testing, human interaction is not required. In automation testing we use tools to run the test cases.
Requirements	To perform manual testing, we need skilled employees, more time and high costs.	To perform automation testing, we require automation tools. Automation testing saves time, cost and manpower. Once test suite is recorded in automation tool then it can easily run test suit.
Application to Test	Any application can be first tested manually. Informal testing like ad-hoc and monkey testing are performed manually.	Automated testing is used to test an application whose requirements are stable at some extent and automation testing mainly used to perform Regression Testing.
Execution of Same test cases	We cannot use manual testing while executing same test suit. Repetitive execution of test cases become boring and error prone.	The most boring part which is of running same test cases repetitively can be performed with the help of automation testing.

6.1.4 Principles of Software Testing

Q. 6.1.4 Explain the principles of software testing. (Ref. sec. 6.1.4) **(10 Marks)**

There are 7 Testing principles which are as follows :

(a) Complete testing of software is not possible

- Complete testing of software is not possible. Rather, we require performing the best possible amount of testing with the help of risk assessment of the application.
- Risk assessment is the process of guessing module where defects may occur and test that module to find out those defects. Because we have not that much amount of time to test each and every module from our software, exhaustive or complete testing of software is not possible.

(b) Defect clustering

- This principle states that about eighty percent of defects are found in twenty percent of modules.
- Using experience, we can find out such risky modules.
- But this approach has its own demerits such as if we run similar test cases repetitively, such test cases does not find any new defect.

Pesticide Paradox

- If we use same test cases repetitively for testing then after particular point we cannot find new defect. To solve this problem, the test cases required to be reviewed and revised on regular basis and include new test cases to help in finding new defects.
- Testers cannot use only existing testing mechanism. He should try to improve the existing methods to find out more defects. But after performing testing, you can never claim your product is bug free.

Testing shows presence of defects

- Testing process can show that the defects are present in the system under test but it cannot prove that there are no defects.
- Even after testing we cannot guarantee that system is 100 % error free.
- Objectives of testing process are as follows :
  - (i) Finding defects which are created by developer while developing software.
  - (ii) Providing information about quality of software under test.
  - (iii) To ensure that system meets the business and the user requirements.
  - (iv) To gain customer's confidence by providing them a qualitative product.

(e) Absence of error

- The software which is 99% bug free can be still unusable if it is tested for wrong requirements i.e. absence of error does not help if system does not satisfy user's need and requirement.

(f) Early Testing

- Testing must be started as early as possible in the Software Development Life Cycle (SDLC).
- Therefore any bug in the requirements or design phase are find out in early phases of software testing.
- It is less costly to correct that bug in early stages of testing.
- It is suggested that we can start detecting the bug at time when requirements are gathered.

(g) Testing is context dependent

'Testing is context dependent' specifies that the way tester test an online shopping site will be different from the way he / she test stand alone application.

6.1.5 Objectives of Software Testing

→ (MU - Dec. 16)

Q. 6.1.5 What are the objective of testing ? (Ref. sec. 6.1.5) **Dec. 16, 5 Marks**

- 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.
- To make sure that it satisfies requirements present in the BRS (Business Requirement Specification) and SRS (System Requirement Specifications).
- 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.

6.1.6 Software Testing Process

Q. 6.1.6 Explain the software testing process in detail. **(10 Marks)**
(Ref. sec. 6.1.6)

- 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.

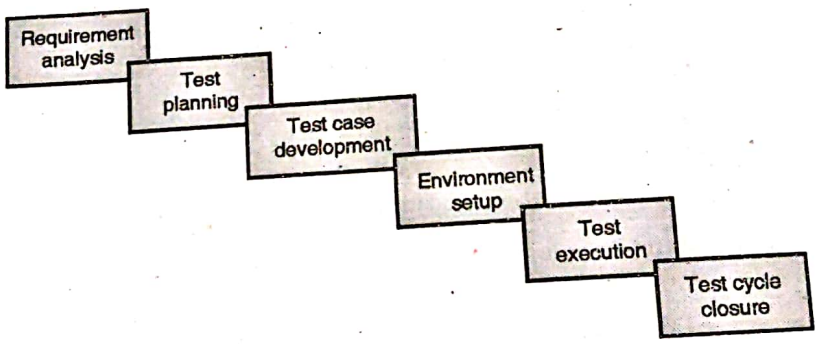


Fig. 6.1.2

- **Entry criteria :** Entry criterions of testing are prerequisite conditions that must be satisfied before testing begins.
- **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 networked if necessary.
- It includes generation, verification and rework (as per requirement) of test cases and test scripts.
- Test cases and test scripts are outcome of this phase.

(iv) Test environment set up

- After completion of set up, smoke test is performed on it.
- Thus ready test environment and result of smoke test are outputs 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.
- Intension of test execution is to find bugs in software.
- These bugs are reported to development team for correcting them.
- After correction retest 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.

Syllabus Topic : Strategic Approach to Software Testing

6.2 Strategic Approach to Software Testing

Q. 6.2.1 Write note on Strategic Approach to software testing. (Ref. sec. 6.2) **(10 Marks)**

- As we have seen, 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 problems should be identified as early as possible.

Syllabus Topic : Verification and Validation

6.2.1 Verification and Validation

Q. 6.2.2 Explain verification and validation in brief. (Ref. sec. 6.2.1) **(5 Marks)**

Verification

- Verification can be defined as 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.
- Now the question is : What is mean by the intermediary products? The answers is the **intermediary products** consists of documents which are generated during the development phases such as requirements specification, design documents, database table design, ER diagrams, test cases, traceability matrix etc.
- Sometimes we can avoid focusing on reviewing these documents but reviewing these documents can help us to find out most of the glitches. If these glitches are found at later phases of the development lifecycle then it will be very costly.

- Verification helps to find out whether the software is of high quality, but it will not guarantee that the system is useful.
- The main focus of the verification is to make the system well-engineered and free from errors.

Validation

- Validation can be defined as it is the process of evaluating the final product to ensure that the software meets all the specified requirements.
- It can also be defined as a process which shows that the product fulfills its intended use when deployed on suitable environment.
- Validation is performed in the software development lifecycle after the verification phase.

6.2.2 Difference between Verification and Validation

Q. 6.2.3 Compare Verification and Validation Testing. (Ref. sec. 6.2.2) → (MU - May 15, Dec. 17) **May 15, Dec. 17, 10 Marks**

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 conducted by quality assurance team.	Validation is conducted 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.

Parameter	Verification	Validation
Processes	Plans, Requirement Specifications, Design Specifications, Code, and Test Cases etc. are evaluated in verification.	Testing of actual software is done in validation.

6.2.3 Software Testing Strategy

Q. 6.2.4 Explain "Software Testing Strategy" with suitable diagram. (Ref. sec. 6.2.3) **(10 Marks)**

- We have already seen phases of software testing process. Now we will see software process with different aspect.
- The software process can be considered as the spiral as shown in Fig. 6.2.1.
- In the beginning, the role of software is defined by the system engineering and proceed to software requirements analysis, in which there is establishment of the information domain, function, behavior, performance, constraints, and validation criteria for software.
- Moving towards inward in the spiral, we come to design and at the end to coding.

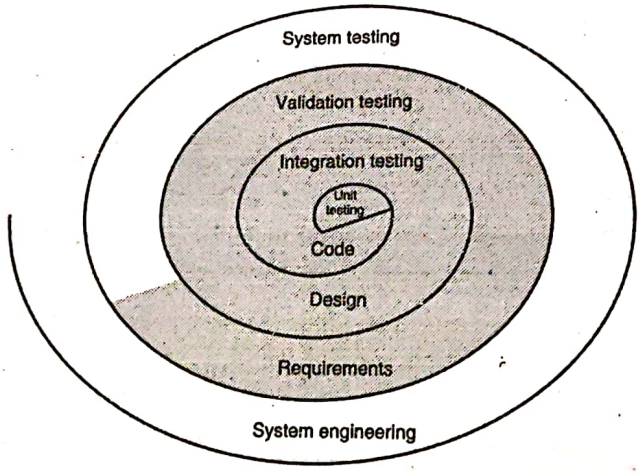


Fig. 6.2.1 : Software Process

- It is possible to view a strategy for software testing in the context of the spiral.
- Unit testing gets started at the vortex of the spiral and focus on all of the software units such as component, class, or WebApp content object which are implemented in source code.
- Testing continues by traversing outward direction along the spiral to integration testing. Here the main concentration is on designing and building the software architecture.
- Taking next turn outward on the spiral, we can observe validation testing, in which requirements which are set as part of requirements modeling are validated against the respective software which has been developed.
- At last we reach at system testing, where testing of the software as well as other system elements is done as a whole.

- For testing any computer software, we have to spiral out in a clockwise direction along streamlines which enhances the scope of testing with each and every turn.
- Taking into account the process from a procedural point of view, in the context of software engineering, practically testing is an implementation of four steps in a sequence.
- The steps are shown in Fig. 6.2.2.
- In the beginning, tests concentrate on every component individually, making sure they function properly as a unit.
- Therefore, the name is unit testing. Unit testing use testing techniques effectively which exercise particular paths in control structure of a component to confirm entire coverage as well maximum possible error detection.

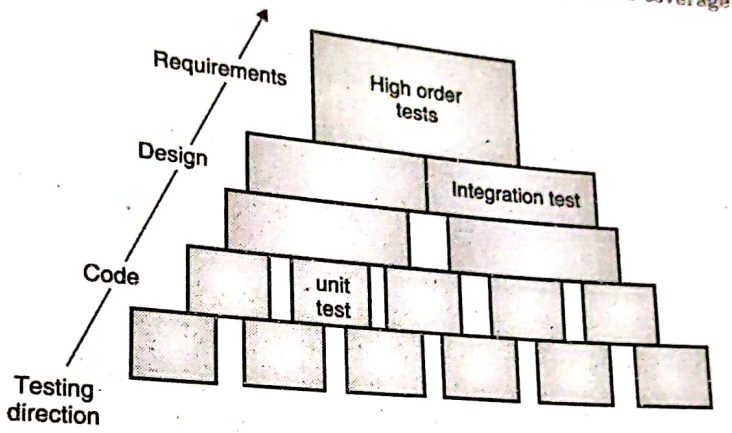


Fig. 6.2.2 : Four Steps of Testing

- Further there is need to assemble or integrate components to form the whole software package.
- Integration testing highlights the concerns which are related to the dual problems of verification and program construction.
- Test case design techniques which in general used to concentrate on inputs and outputs are more common in the process of integration.
- After integration of software, a series of high-order tests is conducted.
- There is need to evaluate validation criteria which has been set during requirements analysis.
- Validation testing presents absolute assurance that all informational, functional, behavioral, and performance requirements are fulfilled by the software.
- The final high-order testing step is considered outside the limit of software engineering and in the wider context of computer system engineering.
- It is necessary to combine the software which is validated with other system elements such as hardware, people, database, etc.
- System testing confirms that all the elements mesh correctly and the overall system function/performance has been achieved.

Syllabus Topic : Unit Testing

6.3 Unit Testing

→ (MU - Dec. 15)

Q. 6.3.1 Explain unit testing. (Ref. sec. 6.3) **Dec. 15. 3 Marks**

- **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.
- We are going to see the Unit testing in detail.

☞ 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

- Unit Test Plan
- Prepare
- Review
- Rework
- Baseline
- Unit Test Cases/Scripts
- Prepare

- Review
- Rework
- Baseline
- Unit Test
- 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.

6.3.1 Stubs and Drivers in Unit Testing

Q. 6.3.2 Draw and explain stub and driver mechanism of unit testing. (Ref. sec. 6.3.2) **(5 Marks)**

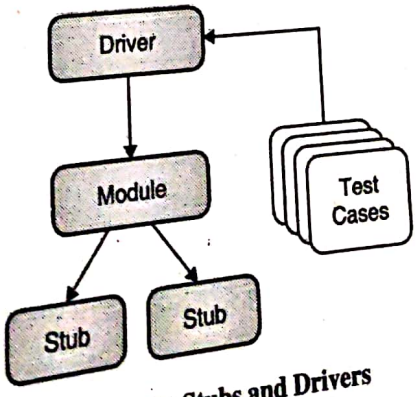


Fig. 6.3.1 : Stubs and Drivers

(1) STUBS

Assume we have 3 modules, Module A, Module B and Module C. Module A is ready and we need to test it, but module A calls functions from Module B and C which are not ready, so developer will write a dummy module which simulates B and C and returns values to module A. This dummy module code is known as **stub.**

(2) DRIVERS

Now suppose we have Modules B and C ready but Module A which calls functions from Module B and C is not ready so developer will write a dummy piece of code for Module A which will return values to Module B and C. This dummy piece of code is known as **driver.**

6.3.2 List of Errors Detected by Unit Testing

Q. 6.3.3 Enlist various types of errors detected by unit testing. (Ref. sec. 6.3.2) **(10 Marks)**

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

- Local data structure
- Boundary conditions
- Independent path
- Error handling path
- Local and global variable
- Incorrect initialization
- Incorrect symbolic representation of an expression
- Incorrect arithmetic precedence
- Global variable naming convention
- Mapping error
- Log and exception handling
- Referred different data type in the code and database
- Necessary input parameter not trim during the saving in the database.

Syllabus Topic : Integration Testing

6.4 Integration Testing

→ **(MU - Dec. 15, Dec. 16)**

Q. 6.4.1 Explain integration testing. (Ref. sec. 6.4) **Dec. 15, Dec. 16, 5 Marks**

- 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.

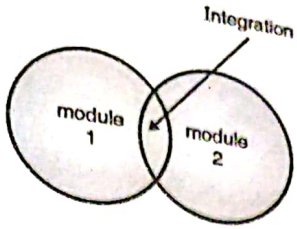


Fig. 6.4.1

Following are reasons for performing Integration testing :

- 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.
- So, while performing Integration Testing, we required to check whether software module works efficiently with other modules or not.
- While developing single module, there is risk of changes in requirements by the clients.
- We may not perform unit testing for this new requirements and so performing integration testing of software is needed.
- There is possibility that interfaces (through which operations on database are done) present between software modules and database may contain error.
- There is possibility of error occurrence due to external hardware interfaces.
- There is possibility of raising issues if exceptions are not handled properly.
- For all these reasons we need to perform integration testing.

6.4.1 Types of Integration Testing

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

(I) Big bang approach

- In this strategy, all modules of software product are created first and then they are combined together and whole software is tested at once.
- Example of big bang approach As per the Fig. 6.4.2 all the modules from 'Module 1' to 'Module 7' are combined one by one and then the integration testing is performed.

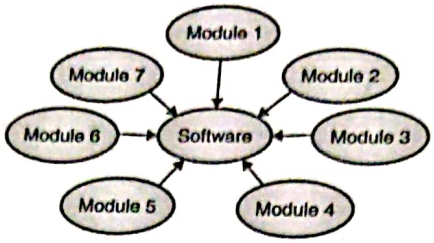


Fig. 6.4.2 : Big bang approach

☞ **Advantage of big bang approach**

- Suitable for small software projects.

☞ **Disadvantages of big bang approach**

- Finding defect is difficult task because we test whole software at once.
- There are lots of interfaces which needed to be test.
- In big bang approach, there is possibility that some interfaces links may remain untested.
- Testing team get small amount of time for performing testing because in this approach integration testing starts after all the modules present in software are developed.
- High risk present in this approach is that critical modules are not separated and tested on priority basis because all modules are tested at the same time.
- Peripheral modules that are related with user interfaces are not separated and tested on priority basis.

(2) Incremental Approach

- In this approach, to perform testing two or more modules are merged with each other which are **logically related.**
- Then other associated modules are included in this group of modules and perform testing for checking whether they functioning correctly or not.
- This procedure continues until all of the modules are grouped together and tested successfully.
- This procedure is done with the help of dummy programs which are knows as **Stubs** and **Drivers.**
- Stubs and Drivers do not contain the complete programming logic of the software module but they contain code which is needed to perform communication with other modules.
- A Stub is dummy program which is called by the Module on which testing is performed. We replace called module by stub.
- Driver is dummy program which calls another module. We replace Module to be tested by driver i.e. Calling module is replaced by driver.

- Incremental Approach is performed using two different Methods such as :
  (a) Top Down (b) Bottom Up

(a) Top down Integration

- In Top to down approach, testing is performed from the modules present at top to modules which are present at the bottom.
- In top down approach, stubs are used for testing.

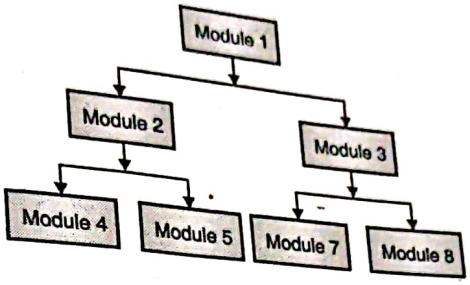


Fig. 6.4.3 : Top down approach

Advantages of Top down approach

- Identification of bug is easy.
- Critical Modules are tested on priority basis so critical designing defects could be found and fixed early.

Disadvantages of Top down approach

- Top down testing requires many stubs because for replacing lower level modules there is need of stubs.
- Modules present at lower level are tested insufficiently because of lack of time.

(b) Bottom up Integration

- In the bottom up approach, every module present at lower level is tested with modules present at higher levels until all modules are tested.
- Drivers are used while performing bottom up incremental testing.

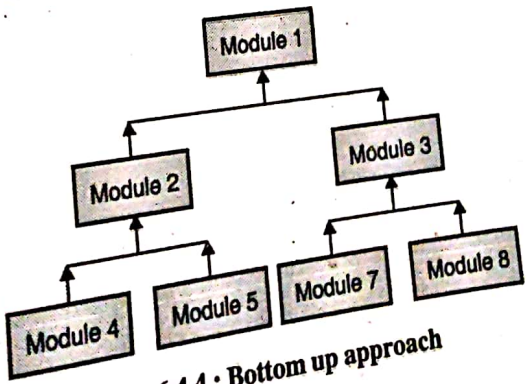


Fig. 6.4.4 : Bottom up approach

Advantages Bottom up approach

- Finding defect is easy task.
- After developing one module, testing is started. It does not waste time in waiting for all modules to be developed unlike Big-bang approach.

Disadvantage Bottom up approach

Critical modules which are present at the top level of software architecture that control the flow of software are tested last and may be defects occurs in critical modules.

Syllabus Topic : Validation Testing

6.5 Validation Testing

Q. 6.5.1 Explain Validation Testing in detail. (Ref. sec. 6.5) **(10 Marks)**

- When integration testing ends, Validation testing begins. After completion of independent components, the software is completely integrated as a package, and interfacing errors have been found out and corrected.
- At the validation or system level, no difference remains in conventional software, object-oriented software, and WebApps.
- Testing is concentrated on the actions which are directly visible by user and output which is user-recognizable from the system.
- There are number of ways to define validation, but a general definition is that validation will be successful when the functionality of software is as per expectations of the customer.
- Here the main concern is user requirements.
- A Software Requirements Specification explains all of the user-visible attributes regarding the software and includes a Validation Criteria section which sets the basis for the approach regarding validation-testing.

6.5.1 Validation-Test Criteria

- To achieve software validation a series of tests is carried out which demonstrates conformity with user requirements.
- A test plan describes the classes regarding tests to be conducted.
- A test procedure defines particular test cases which are designed to ensure following :
  - All of the expected functional requirements are fulfilled.
  - All of the respective behavioral characteristics are achieved.
  - There is accuracy in all the content and are properly presented.
  - There is consideration of all performance requirements.
  - Documentation is accurate.

- Usability as well as other essential requirements are met such as transportability, compatibility, error recovery, maintainability.

After completion of each validation test case, there is one of two possible conditions :

1. There is acceptance of the function or performance characteristic since it conforms to specification or
2. A variation from specification is not covered and a list regarding deficiency is created.

Before scheduled delivery, it is difficult to correct the deviations or errors found at this stage in the project.

There is need to negotiate with the client (customer) to set a method for resolving deficiencies.

6.5.2 Configuration Review

- Configuration review is considered as a vital element of the validation process. The main aim of the review is making sure that all associated elements of the software configuration accurately developed are properly cataloged.
- The configuration review is also known as audit.

6.5.3 Alpha and Beta Testing

Q. 6.5.2 Write note on Alpha and Beta Testing. (Ref. sec. 6.5.3) (10 Marks)

1) Alpha testing

- Alpha Testing is software testing which is done to find out bugs before deploying the software application to end user.
- Alpha testing is a type of acceptance testing.
- This testing is called as an alpha testing since it is performed when development phase of software application is near to Beta Testing.
- The objective of alpha testing is to refine the software product by finding (and fixing) the bugs that were not discovered through previous tests.
- Alpha testing is typically performed by in-house software engineers or QA staff.
- It is the final testing stage before the software is released into the real world.
- Alpha testing is performed in two phases :
  - (i) In first phase software is tested by development team members. They perform debugging of software to catch bugs quickly.
  - (ii) In second phase software is tested by software quality analyst team for additional testing in actual user's environment setup.

Advantages of alpha testing

- Better insight about the software's reliability at its early stages.
- Free up team for other projects.

- Reduce delivery time to market.
- Early feedback helps to improve software quality.

(2) Beta testing

- Beta testing is testing which is performed at the location of customer. In this testing actual as well as intended users will test the software to determine whether the software is satisfying their needs and expectations.
- Beta testing allows users to test software before it is released to public.
- Beta testing minimizes the product failure risks and delivers qualitative product through customer validation. It gives direct feedback from the user.
- It ensures reliability, security, robustness etc. from user's perspective.
- Types of Beta testing are : Traditional, public, technical, focused and past release.
- Beta testing is also called as User Acceptance Testing, Customer acceptance Testing, Customer Validation testing and Pre-Release testing.
- Beta testing gives assurance that we deliver quality software to our customers by testing the product under various situations in real world environment that can't be created in a lab setting.

☞ **Advantages Beta Testing**

- Beta testing decreases product failure risk by performing customer validations.
- Beta Testing permits tester to test the software application in post-launch infrastructure.
- Beta testing improves the quality of software product by using feedback of customer.
- Beta testing is cost effective as compared to other data collection techniques.
- Beta testing helps to increases customer satisfaction.

☞ **Types of beta testing**

There are various types of Beta testing :

1. **Traditional Beta testing :** Software Product is provided to targeted end user and associated data is collected in all aspects. This data is useful for the improvement of Product.
2. **Public Beta Testing :** In this type of beta testing, product is released publicly in real world using online channels and data can be collected from anyone. Using feedback from end users, improvements in product are done.
3. **Technical Beta Testing :** In this type of beta testing, software Product is released in internal group of an organization and collect data from the employees of the organization.
4. **Focused Beta :** In this type of beta testing, software Product is released in the market for collecting feedback about specific features of the program.
5. **Post release Beta :** In this type of beta testing, software Product is released in the market and data is collected to improve the software product for the next release.

6.5.3(A) Difference between Alpha and Beta Testing

Q. 6.5.3 Differentiate between Alpha and Beta Testing. (Ref. sec. 6.5.3(A)) (5 Marks)

Parameter	Alpha Testing	Beta Testing
Performed by	Alpha testing is done by Testers who are generally employees of organization.	Beta testing is done by End Users who are not employees of organization.
Testing environment	Alpha Testing done is in lab environment.	Beta testing is done in real world environment.
Factors tested	Security Testing and Reliability are not tested in depth during Alpha Testing.	Reliability, Security, Robustness is verified during Beta Testing.
Included testing type	Alpha testing includes white box and black box testing.	Beta Testing includes Black Box Testing.
Time required	Alpha testing requires lot of time to perform because in it white box as well as black box testing is performed.	Few weeks of time is required to perform the beta testing.
Fixing of issues	Important issues can be corrected by developers without delay in Alpha testing.	Issues detected in Beta testing will be corrected in future versions of the software product.
Focus	Alpha testing focuses on giving assurance about quality of the software product before going to Beta testing.	Beta testing also focus on quality of the product, but it collects feedback from end users about software product and gives assurance that the software product is ready for use in real world environment.

Syllabus Topic : System Testing

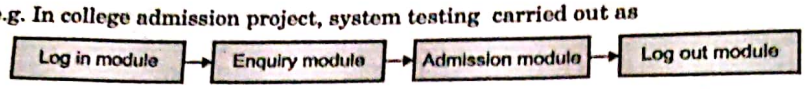
6.6 System Testing

→ (MU - May 17)

Q. 6.6.1 Write short note on System Testing. (Ref. sec. 6.6) May 17, 10 Marks

- 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

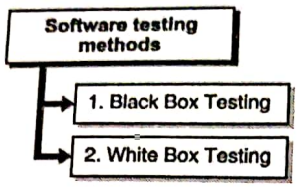


Fig. 6.6.1 : Software testing methods

➜ **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. Below we have discussed some of them :

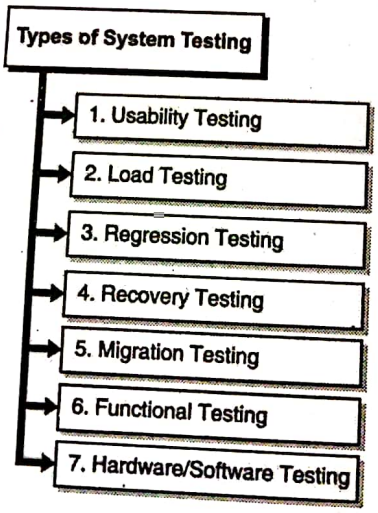


Fig. 6.6.2 : Types of System Testing

➜ ***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".

Syllabus Topic : Software Testing Fundamentals

6.7 Software Testing Fundamentals

- The main aim of testing is to search errors, and a good test is the test which has a high probability of getting an error.
- Hence there is need to design and implement a customized software or a product with keeping "testability" in mind.
- Simultaneously, it is also necessary that the tests themselves should exhibit a set of characteristics which leads to encounter most of the errors with minimum efforts.
- **Testability :** James Bach provides the following definition for testability: "Software testability is simply how easily [a computer program] can be tested."

Following are characteristics of a good testable software :

- **Operability :** "The better it works, the more efficiently it can be tested." If quality is focused while designing and implementing a system, comparatively small number of bugs will block the execution of tests, allowing smooth testing.

- **Observability :** "What you see is what you test." Inputs which are given as part of testing return distinct outputs. System states as well as variables can be seen or queriable through execution. Wrong output is simply identified. Detection and reporting of internal errors is done automatically. Source code is accessible.
- **Controllability :** "The better we can control the software, the more the testing can be automated and optimized." Generations of all possible outputs is done using some combination of input, and I/O formats. The entire code can be executed by some combination of input. It is possible to control software and hardware states and variables straightly by the test engineer. It is also possible to specify tests conveniently, automate them, and reproduced.
- **Decomposability :** "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting." The development of software system is implemented from individual modules which can be tested independently.
- **Simplicity :** "The less there is to test, the more quickly we can test it." Functional simplicity is expected from the program (e.g., the feature set is the minimum necessary to meet requirements); structural simplicity (e.g., architecture is modularized to limit the propagation of faults), and code simplicity (e.g., a coding standard is adopted for ease of inspection and maintenance).
- **Stability :** "The fewer the changes, the fewer the disruptions to testing." If Modifications to the software are less then it is easy to control them and the recovery of software is fast from failures.
- **Understandability :** "The more information we have, the smarter we will test." If there is availability of thorough information regarding architectural design and the dependencies among internal, external, as well as shared, test will be smarter.

6.7.1 Characteristics of a Good Test

Q. 6.7.1 Write Characteristics of a Good Test. (Ref. sec. 6.7.1) (5 Marks)

There are some characteristics of a good test as follows :

- *There is high probability of finding out errors in a good test.* To achieve this goal, there is need for the tester to understand the software thoroughly and try to develop a view regarding how the software might fail.
- Preferably, the classes regarding failure are searched. For example, one class of possible failure in a GUI is the failure to identify proper mouse position.
- A series of tests possibly developed to exercise the mouse in an effort to exhibit an error in mouse position recognition.
- *A good test is not redundant.* There is limitation in testing time as well as resources.
- There is no meaning in performing a test which has same intention as another test.
- The purposes of different tests must be different.
- *A good test should be "best of breed".* In a set of tests which have same aim, time as well as resource limitations may mitigate toward the implementation of just a subset of these tests.
- In such situation, one should prefer the test having highest possibility of uncovering a whole class of errors.

- *A good test should be neither too simple nor too complex.* Even if sometimes tester can combine a series of tests into one test case, he should remember that the possible side effects related with this approach may mask errors.
- Usually it is better to execute each test separately.

Syllabus Topic : White-Box Testing

6.8 White-Box Testing

Q. 6.8.1 Write note on White Box Testing. (Ref. sec. 6.8) **(10 Marks)**

- White Box Testing is testing process in which we verify internal coding and infrastructure of software product under the test.
- White box testing is based on analysing the internal implementation of system under test. Thus programming knowledge or detailed functional knowledge of system is major pre-requisite for tester.
- White box testing mainly concentrates on the testing flow of inputs and outputs through the software, strengthening security and improving design and usability of software product.
- White box testing is also called as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing.
- The term "white box" is used to indicate testing of code of software because we can see what is present inside the box.
- The clear box or white box term indicate the capability to see via outer part of software into its internal workings.
- It uses following methods :
  - **Statement coverage** i.e. testing all programming statements using minimum number of tests.
  - **Branch coverage :** This is to ensure that all branch conditions in system are tested at least once.
  - **Path coverage :** It includes ensuring that each statement and branch in system is tested at least once.
- Other than above there are different coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc.
- Every technique has its own advantages and tries to cover all parts of software's product code.
- By using Statement and Branch coverage, we can perform 80-90% code coverage which is enough.

White box testing involves two steps. They are :

***Step 1 :* Understand the source code**

- In this step, testers understand the source code of the software product.

- White box testing includes testing of internal working of software product. So the tester must have very high level knowledge of programming languages used in the software product under the test.
- Security is basic purpose of performing the software testing. So the tester should be able detect security problems and protect the software from attacks by hackers and naive users who may add malicious code in software product knowingly or unknowingly.

Step 2 : **Create test cases and execute**

- The second step involved in white box testing includes testing the source code of software product under the test for checking of proper flow of control and structure.
- The tester will generate small test cases for every process or group of processes in the software product.
- This method requires that the tester should have high level knowledge about source code.
- White box testing may be performed by the developers because developers have high level knowledge of programming language used in software product under the test.

6.8.1 Advantages of White Box Testing

Following are the advantages of white box testing :

- As it tests all possible paths of system it is a through testing.
- White Box Testing can be applied at earlier testing stage as there is no need to wait for GUI.
- As tester has knowledge of internal coding, it becomes easy for tester to find out which type of input helps is testing.
- White box testing performs Code optimization through finding hidden bugs.
- Automation of White box tests cases can be done easily.
- White box testing is strict because each and every statement of code tested at least one time.
- We can start white box testing early as soon as first module of software is created.

6.8.2 Disadvantages of White Box Testing

Following are the disadvantages of white box testing :

- White Box Testing is complex and expensive process.
- For testing larger applications using White Box Testing, it becomes impossible to perform exhaustive testing.
- White Box Testing tests the software as it exists, hence missing functionality of system may not be discovered.
- It is necessary to select all possible inputs to test each path which is time consuming process.
- If white box testing performed by developers who develop code present in software then there is less possibility to find out development related errors.
- While performing White box testing, we need expert persons who have deep knowledge of programming language in which Application Under Test (AUT) builds.

Syllabus Topic : Basis Path Testing

6.9 Basis Path Testing

→ (MU - Dec. 15, Dec. 17)

Q. 6.9.1	Explain different techniques in white box testing. (Ref. sec. 6.9)	Dec. 15, 10 Marks
Q. 6.9.2	Explain Basis Path of Testing. (Ref. sec. 6.9)	Dec. 17, 5 Marks

- 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.

6.9.1 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.

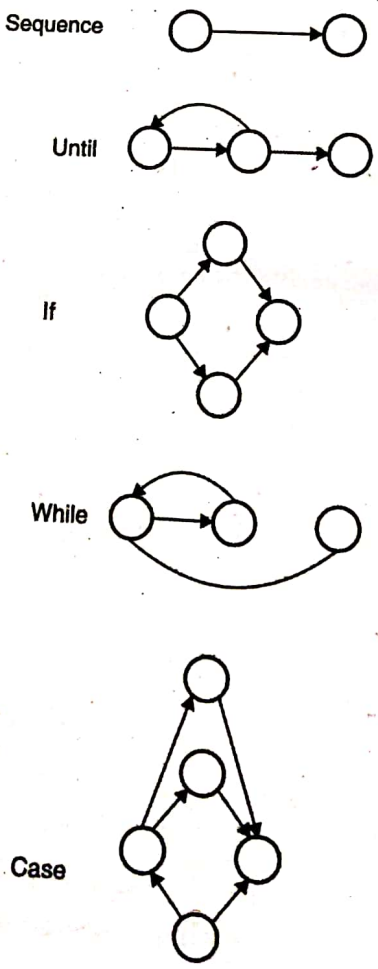


Fig. 6.9.1

6.9.2 Basic Terminology associated with The Flow Graph

- **Node :** Each flow graph node represents one or more procedural statements. Each node that contains a condition is called a predicate node.
- **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.
- **Region :** A region in a flow graph is an area bounded by edges and nodes.

6.9.3 Cyclomatic Complexity

Q. 6.9.3 Explain cyclomatic complexity with suitable example. (Ref. sec. 6.9.3) → (MU - Dec. 17, May 18) **Dec. 17, May 18. 5 Marks**

- 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.

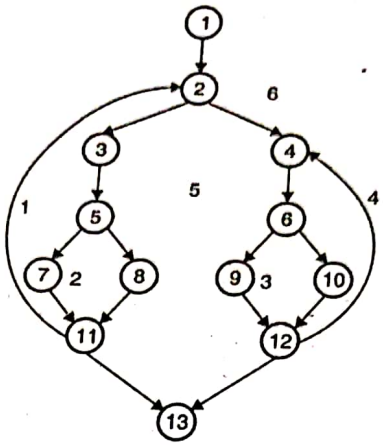


Fig. 6.9.2

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

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

$= 17 - 13 + 2 = 6$

6.9.4 Deriving Test Cases

- The main objective of basic path testing is to derive the test cases for the procedure under test. The process of deriving test cases is as follows :

From the design or source code, derive a flow graph.

1. Determine the Cyclomatic complexity, V(G) of this flow graph using any of the formula discussed above.
2. Even without a flow graph, V(G) can be determined by counting the number of conditional statements in the code and adding one to it.

3. Prepare test cases that will force execution of each path in the basis set. Each test case is executed and compared to the expected results.

6.9.5 Graph Matrices

- Graph matrix is a two dimensional matrix that helps in determining the basic set. It has rows and columns each equal to number of nodes in flow graph.
- Entry corresponding to each node-node pair represents an edge in flow graph.
- Each edge is represented by some letter to distinguish it from other edges.
- Then each edge is provided with some link weight, 0 if there is no connection and 1 if there is connection.
- For providing weights each letter is replaced by 1 indicating a connection.
- Now the graph matrix is called connection matrix.
- Each row with two entries represents a predicate node.
- Then for each row sum of the entries is obtained and 1 is subtracted from it.
- Now the value so obtained for each row is added and 1 is again added to get the cyclomatic complexity.
- Once the internal working of the different procedures is tested, then the testing for the overall functionality of program structure is tested. For this black box testing techniques are used.

Syllabus Topic : Control Structure Testing

6.10 Control Structure Testing

Q. 6.10.1 Explain Control Structure Testing in detail. (Ref. sec. 6.10) **(10 Marks)**

- The basis path testing technique which we have seen in previous section is itself a technique which comes under the category control structure testing.
- Even if basis path testing is considered as simple as well as highly effective, it is insufficient in itself.
- Now we are going to see some other variations on control structure testing.
- These broaden testing helps to improve the quality of white-box testing.

6.10.1 Condition Testing

- Condition testing is method regarding a test-case design which workout the logical conditions existing in a program module.
- A simple condition is a Boolean variable or a relational expression, which most probably preceded with one NOT (¬) operator.

- A relational expression takes the form

E1 <relational-operator> E2

where E1 and E2 are arithmetic expressions whereas the <relational-operator> is any one of >, >=, =, #(non-equality), < or <=.

- In a compound condition elements present are; two or more simple conditions, Boolean operators, and parentheses.
- It is considered that Boolean operators which are probably permitted in a compound condition contains OR (|), AND (&), and NOT (¬).
- A condition which does not contain relational expressions is considered as a Boolean expression.
- If a condition is wrong, then minimum single component of the condition is wrong. Hence, types of errors in a condition contain Boolean operator errors (incorrect/missing/extra Boolean operators), Boolean variable errors, Boolean parenthesis errors, relational operator errors, and arithmetic expression errors.
- The focus of condition testing is mainly on testing each and every condition in the application to make sure that it is error free.

6.10.2 Data Flow Testing

- The data flow testing technique opts for test paths of a program based on the placements of definitions and uses of variables in the respective program.
- For the illustration of the data flow testing approach, consider that a unique statement number is assigned to every statement in a program. Also we will assume that functions in the program will not make any changes in their parameters or global variables.
- For a statement with S as its statement number,

DEF(S) {X | statement S contains a definition of X} USE(S) {X | statement S contains a use of X}

- If statement S is an if or loop statement, its DEF set does not contain anything i.e. empty and its USE set depends upon the condition of statement S.
- The definition of variable X at statement S can be considered as live at the location of statement S' provided a path is present from statement S to statement S' which do not have any other definition of X.
- A DU (definition-use) chain of variable X is of the form [X, S, S'], where S and S' are considered as the statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is live at statement S'.
- An easy strategy regarding data flow testing is to need that each DU chain be covered minimum once. This strategy is known as DU testing strategy.
- Usually it is seen that that DU testing does not assure the coverage of each and every branch of a program.
- Yet, a branch is not sure to be covered by DU testing merely in exceptional situations like if-then-else constructs in which there is no definition of any variable in *then* part and the *else* part.

- In such case, the else branch of the respective if statement is not essentially covered by DU testing.

6.10.3 Loop Testing

- Loops are considered as the foundation stone for maximum of the algorithms implemented in software.
- However while conducting software tests, they are not given sufficient importance.
- Loop testing is considered as a white-box testing technique which focuses completely on the validity of loop constructs.
- For loops it is possible to define four different classes: simple loops, concatenated loops, nested loops, and unstructured loops.

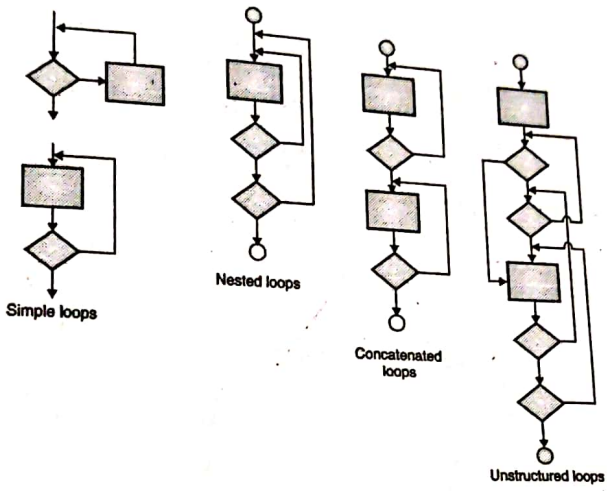


Fig. 6.10.1 : Classes of Loops

- **Simple loops :** The following given series of tests can be carried out on simple loops, where n is the maximum number of permissible passes through the loop.
  1. Skip the loop entirely.
  2. Only one pass through the loop.
  3. Two passes through the loop.
  4. m passes through the loop where $m < n$.
  5. $n - 1$, n, $n + 1$ passes through the loop.
- **Nested loops :** If there is need to extend the test approach for simple loops to nested loops, there is increase in the number of possible tests with increase in the level of nesting.
- This would leads to execute impractical number of tests. Beizer (an American software engineer) suggests an approach which helps to decrease the number of tests :
  1. Begins with the innermost loop. Set minimum values to all other loops.

2. Carry out easy loop tests for the innermost loop when keeping the outer loops at their lowest iteration parameter (for example loop counter) values. Introduce different tests for out-of-range or excluded values.
3. Work outward, while executing tests for the next loop, but holding all other outer loops at minimum values whereas other nested loops to "typical" values.
4. Continue the process unless all loops have been tested.

- **Concatenated loops :** It is possible to test concatenated loops with the help of the approach defined for simple loops, if all of the loops are not interdependent.
- However, if there is concatenation of two loops and the loop counter for first loop is used as the starting value for second loop, then the loops are not independent.
- When there is dependency of loops on each other, then the approach which has been applied to nested loops is recommended.
- **Unstructured loops :** Whenever possible, it is essential to redesign this class of loops so as to reflect the use of the structured programming constructs.

Syllabus Topic : Black Box Testing

6.11 Black Box Testing

→ (MU - Dec. 16)

Q. 6.11.1 Explain Black Box testing. (Ref. sec. 6.11) **Dec. 16, 5 Marks**

- 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.

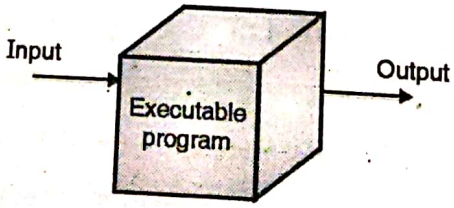


Fig. 6.11.1 : Black box testing

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.

6.11.1 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.

6.11.2 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.

6.11.3 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.

6.11.4 Difference between White Box Testing and Black Box Testing Technique

→ (MU - May 15, May 16)

Q. 6.11.2 Compare white box testing and black box testing. (Ref. sec. 6.11.4) **May 15, May 16, 9 Marks**

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.

Syllabus Topic : Software Maintenance

6.12 Software Maintenance

→ (MU - May 17)

Q. 6.12.1 Write short note on Software Maintenance. (Ref. sec. 6.12) **May 17, 10 Marks**

- 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.

6.12.1 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.

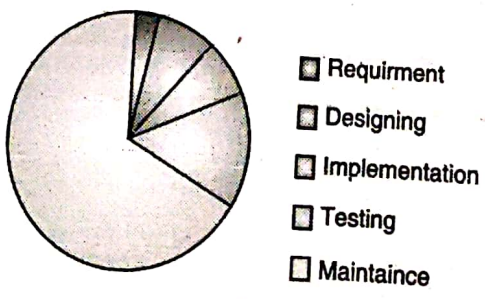


Fig. 6.12.1 : Cost of maintenance

- 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**

- Structure of source code of software.
- Programming Language.
- External environment dependencies.
- Staff availability and reliability.

Syllabus Topic : Types of Software Maintenance

6.13 Types of Software Maintenance

→ (MU - May 16, Dec. 16)

Q. 6.13.1 What are the different types of maintenance? (Ref. sec. 6.13) **May 16, Dec. 16, 5 Marks**

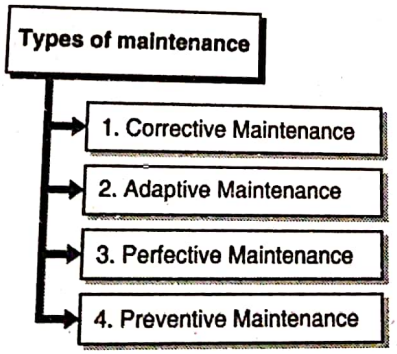


Fig. 6.13.1 : Types of maintenance

1. Corrective Maintenance

- This type of maintenance contains changes and updation 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.
- We can also say that 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.

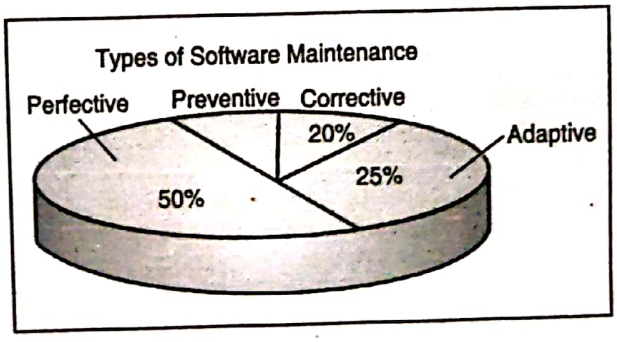


Fig. 6.13.2

6.13.1 Steps to Create Maintenance Log

→ (MU - May 16, Dec. 16)

Q. 6.13.2 Explain steps for creating maintenance log. (Ref. sec. 6.13.1) **May 16, Dec. 16, 5 Marks**

- A maintenance log is a document that records who did what, when, and why.
- Maintenance logs are extremely useful for troubleshooting recurring or obscure problems, as they provide a record of all work performed on the system and may shed light on hard-to-spot interactions between seemingly unrelated symptoms.
- Maintenance of your valued assets is an essential chore to perform at regular intervals. What would happen if you own a precious metal ornament and it gets rusty over time? It will certainly lose its value, therefore to keep your assets in a presentable and functional condition you need to maintain them on a timely basis. Same is the case with your property and vehicle that if they are not maintained with time they will be de-valued.
- To keep an organized maintenance program it is better to design a log in which you can put your maintenance schedule of every equipment you own.

Let us consider the example :

Equipment Maintenance Log

Name of Equipment		**Manufacturer's contact details:**	
Label		**Date of purchase:**	10/15/2016
Serial number:		**Person responsible for equipment:**	
Manufacturer:		**Date put into service:**	10/23/2016

Date:	Maintenance Description	Maintenance performed by:	Date of validation before put into service:	Validation performed by:	Next maintenance planned on (date):	Remarks:

- The maintenance log is a thing which helps you in a case of collapses, damages and repairs. Maintenance logs are crucial because they help to maintain the internal or external scam of your set program, stuff, kits or vehicle on periodic interludes.

- The maintenance log is usually sketched in the contract with the companies; these companies provide services on contract basis.
- Generally, maintenance logs refer to get any preventive measure on your property, equipment or services that it will retain its worth in future.
- The equipment maintenance log compilation will aid you in the following areas. You can :
  1. Design your program knowing that the maintenance task is near.
  2. Make the resources ready for maintenance.
  3. Outline the cost of maintenance.
  4. Smooth flow of operations with no bugs
  5. Emergencies are possible with a maintained environment.
- Several pre-designed on-line templates are available for sketching your maintenance logs. Many websites offer different categories for the different maintenance program. Microsoft excel also documents this log in a user-friendly way. Just download the template and maintain everything.

Syllabus Topic : Software Re-engineering

6.14 Software Re-engineering

Q. 6.14.1 Write note on Software Re-engineering. (Ref. sec. 6.14) **(10 Marks)**

- When we required to update the software application without affecting its functionality so that it can stay in the current market, it is known as software re-engineering.
- Software Re-Engineering is process of re-structuring or re-writing components of old software application without modifying its functionality.
- It is a procedure where the design of software is modified and source code is re-written.
- Old software system cannot use with the latest technology available in the market. If we use it to perform some task, then we may face lots of problems such as hardware become out of date, updating of software becomes a problem.
- For example, in the beginning UNIX was developed using assembly language. When C language was developed, UNIX was re-engineered in C, as writing source code in assembly language and understanding it become difficult.
- Sometimes developers observe that some components of software product require more maintenance than other components and such components are requiring re-engineering.

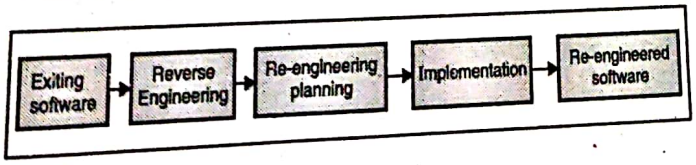


Fig. 6.14.1

- Re-Engineering Procedure includes following steps :
  1. **Decide** which components of software we want to re-engineer. Is it complete software or some components of it ?
  2. **Do** Reverse Engineering to find out features of existing software.
  3. **Perform restructuring of source code** if needed. For example modifying function-oriented programs in object-oriented programs.
  4. **Perform restructuring of data** if needed.
  5. **Use Forward engineering** idea to generate re-engineered software.
- There are some important terms used in Software re-engineering.

1. 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.

Fig. 6.14.2 : Reverse Engineering

2. Restructuring of source code

- Restructuring of source code is a procedure of performing the re-structuring and re-construction of the already existing software.
- It is associated with re-arranging the source code. In Restructuring of source code, we can perform source code-restructuring or data-restructuring or both.
- Re-structuring does not affect the functionality of the existing software. It improves reliability and maintainability of software.
- Program parts due to which errors occur frequently can be changed, or updated in re-structuring.
- The dependency of software on hardware platform can be decreased with the help of re-structuring.

3. Forward Engineering

- Forward engineering is a procedure of finding desired software product from the requirements in hand which is output of reverse engineering.
- It presumes that some software engineering is done in the past.
- Forward engineering is similar as software engineering procedure with having one difference that it is always performed after reverse engineering.

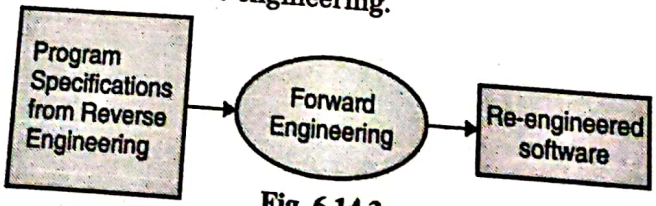


Fig. 6.14.3

Component reusability

- A component is an element of source code of software which performs an independent task in the software system.
- It may be a small module or sub-system.

Syllabus Topic : Reverse Engineering

6.15 Reverse Engineering

→ (MU - May 15, May 16, May 18)

Q. 6.15.1 Explain software reverse engineering in detail. (Ref. sec. 6.15) **May 15, May 18, 10 Marks**

Q. 6.15.2 Write note on Reverse Engineering. (Ref. sec. 6.15) **May 16, 10 Marks**

- 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.

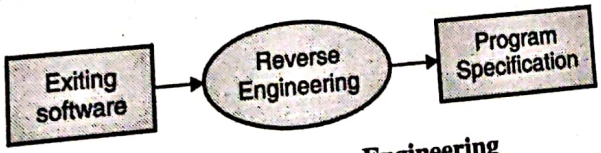


Fig. 6.15.1 : Reverse Engineering

Chapter Ends...

❑❑❑