MASTER OF COMPUTER APPLICATION

MCA-14

SOFTWARE PROGRAMMING

Directorate of Distance Education
Guru Jambheshwar University of Science & Technology, Hisar – 125001

CONTENTS

SUBJECT: Software Engineering	
COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 1	

Introduction to Software Engineering

STRUCTURE

1.0 Learning Objectives

The main objective of this lesson is to lean define software engineering, Importance of software engineering to solve software crisis, Major stages required to develop a software systemic pattern.

1.1 Basic definitions

1.1.1 Program

A program is a set of instructions (written in form of human-readable code) that performs a specific task.

1.1.2 Software

I The **software** is a collection of integrated programs.

II Software is defined as a collection of programs, procedures, rules, data and associated documentation. The software is developed keeping in mind certain hardware and operating system consideration commonly known as platform and software engineering means systematic procedure to develop software. Some of the software characteristics are, it can be engineer or developed and second thing is software is complex in nature.

III Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

IV Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Types of software:- Computer s/w is mainly divided into two types.

a) System software Application s/w consists of programs to perform user oriented tasks. System s/w includes the operating system & all the utilities to enable the computer to run. Ex-window operating system

b) Application software. Application s/w consists of programs to perform user oriented tasks. Ex-word processor, database management. Application s/w sits about the system s/w because it needs help of the system s/w to run.

1.1.3 Engineering

I Engineering is all about developing products, using well-defined, scientific principles and methods.

II Engineering is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc

1.2 Software Crisis

In the late 1960s, it became clear that the development of software is different from manufacturing other products. This is because employing more manpower (programmers) later in the software development does not always help speed up the development process. Instead, sometimes it may have negative impacts like delay in achieving the scheduled targets, degradation of software quality, etc. Though software has been an important element of many systems since a long time, developing software within a certain schedule and maintaining its quality is still difficult.

The term software crisis revolves around three concepts: complexity, change and the expectations. This term was given by F. L. Bauer at the first NATO Software Engineering Conference in 1968 at Garmisch, Germany. Current System design approach is exceedingly empirical. It is unable to cope with increasing systems complexity. A number of problems in software development were identified in 1960s, 1970s, and 1980s. The problems that software projects encountered were: the projects ran over-budget, caused damage to property even to life. Despite our rapid progress, the software industry is considered by many to be in a crisis. Some 40 years ago, the term "Software Crisis" emerged to describe the software industry's inability to provide customers with high quality products on schedule.

Software crisis is also referred to the inability to hire enough qualified programmers. It has become the longest continuing "crisis" in the engineering world and it continues unabated. The most visible symptoms of the software crisis are late delivery, over budget; Product does not meet specified requirements, inadequate documentation. One of the most serious complaints against software failure is the inability to estimate with acceptable accuracy the cost, resources, and schedule necessary for a software project. Conventional assessment methods have always produced positive results which contribute to the too well-known cost infested and schedule slippage.

History has seen that delivering software after the scheduled date or with errors has caused large scale financial losses as well as inconvenience to many. Disasters such as the Y2Kproblem affected economic, political, and administrative systems of various countries around the world. This situation, where catastrophic failures have occurred, is known as **software crisis.**

To explain the present software crisis in simple words, consider the following. The expenses that organizations all around the world are incurring on software purchases compared to those on hardware purchases have been showing a worrying trend over the years. Not only are the software products turning out to be more expensive than hardware, but they also present a host of other problems to the

customers: software products are difficult to alter, debug, and enhance; use resources no optimally; often fail to meet the user requirements; are far from being reliable; frequently crash; and are often delivered late. Among these, the trend of increasing software costs is probably the most important symptom of the present software crisis.

In brief , we can summarize Software Crisis as:

- It was in the late 1960s when many software projects failed.
- Many software became over budget. Output was unreliable software which is expensive to maintain.
- Larger software was difficult and quite expensive to maintain.
- Lots of software not able to satisfy the growing requirements of the customer.
- Complexities of software projects increased whenever its hardware capability increased.
- Demand for new software increased faster compared with the ability to generate new software.

1.2.1 Causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software
- At that time Projects was running over-time
- At that time Software was very inefficient
- The quality of software was low quality
- Software often did not meet requirements
- The average software project overshoots its schedule by half
- At that time Software was never delivered

1.2.2 Solution of Software Crisis:

There is no single solution to the crisis.one possible solution of software crisis is Software Engineering because software engineering is a systematic, disciplined and quantifiable approach. These engineering models helped companies to streamline operations and deliver software meeting customer requirements.

- The late 1970s saw the widespread uses of software engineering principles.

- In the 1980s saw the automation of software engineering process and growth of (CASE) Computer-Aided Software Engineering.
- The 1990s have seen an increased emphasis on the 'management' aspects of projects standard of quality and processes just like ISO 9001

For preventing software crisis, there are some guidelines:

- Reduction in software over-budget
- The quality of software must be high
- Less time needed for software project
- Experience working team member on software project
- Software must be delivered

1.3 Software Engineering

The term software engineering is the product of two words, software, and engineering.

Definitions of Software Engineering

I Software Engineering is a systematic approach to the design, development, operation, and maintenance of a software system.

II Software engineering is an engineering discipline that's applied to the development of software in a systematic approach (called a software process).

II Software engineering stands for the term is made of two words, Software and Engineering. Software is more than just a program code. A program is an executable code, which serves some computational purpose.

III Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IV IEEE definition of Software engineering:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

V Fritz Bauer, a German computer scientist, defines software engineering as: Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

VI Application of engineering for development of software is known as software engineering. It is the systematic, innovative technique and cost effective approach to develop software.

1.3.1 Person involved in developing product is called software engineer. Software engineer is a licensed professional engineer who is skilled in engineering discipline.

Qualities / Skills possessed by a good software engineer:

a. **General Skill** (Analytical skill, Problem solving skill, Group work skill)
b. **Programming Skill** (Programming language, Data structure , Algorithm , Tools(Compiler, Debugger))
c. **Communication skill** (Verbal, Written, Presentation)
d. **Design Skill** (software engineer must be familiar with several application domain).

1.3.2 Software Engineering Processes:-

Process: The process is a series of states that involves activities, constraints, resources that produce an intended output of some kind. Or it is a state that takes some input and produced output Software process: A s/w process is a related set of activities & sub processes that are involved in developing & involving a software system. There are four fundamental process activities carried out by software engineering while executing the software process is:

i) Software specification: The functionality of software & constraints on its operation must be defined.

ii) Software development: The software that mixes the specification must be produced.

iii) Software validation: The software must be validated to ensure that it performs desired customer activities.

iv) Software evolution: The software must evolve to meet changing customer needs with time. The scenario of s/w process & its sub process are shown below:-

The software industry considers the entire software development task as a process according to Booch & Rumbaugh. According to them a process defines who is doing what, when & how to reach a certain goal.

1.3.3 Characteristic of a software process:

1. Understandability
2. Visibility
3. Reliability
4. Robustness
5. Adaptability
6. Rapidity
7. Maintainability
8. Supportability

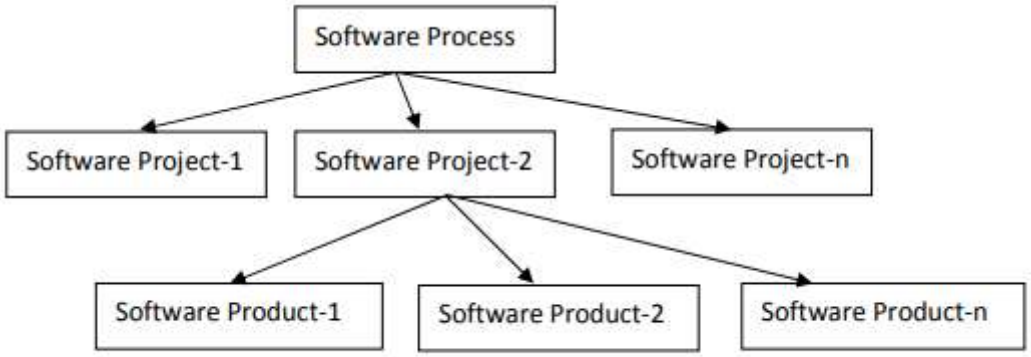
1.3.4 Software Project:

1. A project is a temporary endeavour undertaken to create a unique product.

2. Temporary means every project has a definite beginning and a definite end.

3. Unique means the product must be different in some ways from all similar products.

1.3.5 Software Product: Outcome of software project is known as software product. Software process vs software project vs software product



1.1 Types of Software Product

Based on how and why the software products are being developed, software products can be of the following two types:

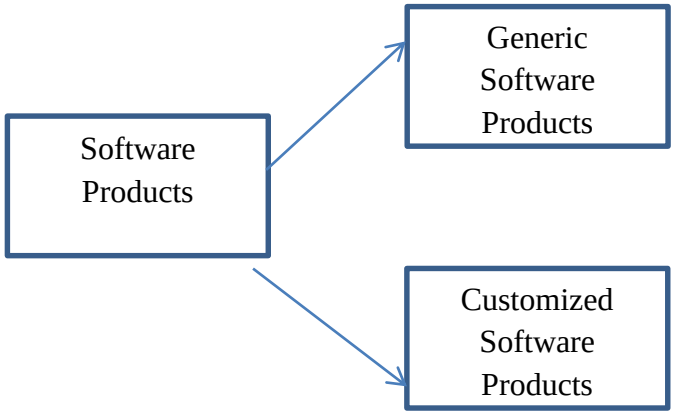


Figure 1.2: Types of Software Product

a) Generic Software Products

Software products which were developed with the target to sell them to customer eligible to buy with no customization for any specific customer are called generic software products. These software products are stand-alone, can be up-gradable to new versions or updates while the updates are prepared by the software company or vendor who developed the product. Developer or Software Developer Company develops the software as it will be used by end users from various levels such as beginner, non-technical etc. and the design the interface of the software and workflow of the software in a way so these end-users from various levels can all get used it easily. Thus generic software products are software products with a large number of end users from various level of experience as target.

b) Customized Software Products

Software products which were developed as per the requirements of a specific customer. Customized software products means a piece of software customized in case of features, workflow, design, language etc. or with additional features included or removed features and financed by a customer as the development cost or price of the software being developed. Customized software can be developed from scratch such as gathering all the requirements from the customer, analyzing and developing using

various software development process models or it can also be that a software product already exists and it will modified as per the requirements of the customer.

1.3.6 Difference between Program and Software product

Table 1.1 illustrate the difference between Program and Software product

Table 1.1 Program vs Software product

Program	**Software product**
Set of instruction related each other.	Collection of program designed for specific task.
Programs are defined by individuals for their personal use.	A sum product is usually developed by a group of engineers working as a team.
Usually small size.	Usually large size.
Single user.	Large no of users.
Single developer.	Team of developer.
Lack proper documentation.	Good documentation support.
ADHOC development.	Systematic development.
Lack of UI.	Good UI.
Have limited functionality.	Exhibit more functionality.

1.3.7 Factor in emergence of software engineering:

1. People who are developing software were consistently wrong in their estimation of time, effort and cost.

2. Reliability and maintainability was difficult of achieved

3. Fixing bug in delivered software was difficult

4. Delivered software frequently didn't work

5. Changes in ration of how to software cost

6. Increased cost of software maintenance

7. Increased demand of software

8. Increased demand for large and more complex software system 9. Increasing size of software

1.4 Objectives of Software Engineering:

a) Maintainability –

It should be feasible for the software to evolve to meet changing requirements.

b) Correctness –

A software product is correct, if the different requirements as specified in the SRS document have been correctly implemented.

c) Reusability –

A software product has good reusability, if the different modules of the product can easily be reused to develop new products.

d) Testability –

Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.

e) Reliability –

It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.

f) Portability –

In this case, software can be transferred from one computer system or environment to another.

g) Adaptability –

In this case, software allows differing system constraints and user needs to be satisfied by making changes to the software.

1.5 Software Characteristics

As we know that software is any computer program which can also be defined as a set of instructions which are responsible for guiding the computer to perform certain tasks. The following are the

characteristics of software:

a) **Software does not wear out**
b) **Software is not manufacture**
c) **Usability of Software**
d) **Reusability of components**
e) **Flexibility of software**
f) **Maintainability of software**
g) **Portability of software**
h) **Reliability of Software**
i) **Software doesnot wear out**

a) **Software does not wear out**:

Different things like clothes, shoes, ornaments do wear out after some time. But, software once created never wears out. It can be used for as long as needed and in case of need for any updating, required changes can be made in the same software and then it can be used further with updated features.

b) **Software is not manufactured**:

Software is not manufactured but is developed. So, it does not require any raw material for its development.

c) **Usability of Software**:

The usability of the software is the simplicity of the software in terms of the user. The easier the software is to use for the user, the more is the usability of the software as more number of people will now be able to use it and also due to the ease will use it more willingly.

d) Reusability of Software

As the software never wears out, neither do its components, i.e. code segments. So, if any particular segment of code is required in some other software, we can reuse the existing code form the software in which it is already present. This reduced our work and also saves time and money.

e) **Flexibility of software**:

A software is flexible. What this means is that we can make necessary changes in our software in the future according to the need of that time and then can use the same software then also.

f) **Maintainability of software**:

Every software is maintainable. This means that if any errors or bugs appear in the software, then they can be fixed.

g) **Portability of software**:

Portability of the software means that we can transfer our software from one platform to another that too with ease. Due to this, the sharing of the software among the developers and other members can be done flexibly.

h) **Reliability of Software**:

This is the ability of the software to provide the desired functionalities under every condition. This means that our software should work properly in each condition.

1.6 Software Engineering Principles:-

Software engineering is a layered technology. The bedrock that supports software engineering is a quality focus. The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work product (models, documents, data, reports, forms, etc.) are produced, milestones are

established, quality is ensured, and change is properly managed. Software engineering methods provide the technical how-to's for building software. That encompasses requirements analysis, design, program construction, testing, and support.

Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques. Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering (CASE), is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

1.7 Evolution of Software Engineering

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information. Software transforms personal data (e.g., an individual’s financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms. The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. The lone programmer of an earlier era has been replaced by a team of software specialists,

each focusing on one part of the technology required to deliver a complex application. And yet, the same questions asked of the lone programmer are being asked when modern computer-based systems are built:

1) Why does it take so long to get software finished?
2) Why are development costs so high?
3) Why can't we find all the errors before we give the software to customers?
4) Why do we continue to have difficulty in measuring progress as software is being developed?

1.8 Software Development Life Cycle Phases

Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying and maintaining the software.

SDLC defines the complete cycle of development i.e. all the tasks involved in planning, creating, testing, and deploying a Software Product.

Purpose: Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement.

SDLC has defined its phases as, Requirement gathering, Designing, Coding, Testing, and Maintenance. It is important to adhere to the phases to provide the Product in a systematic manner. For Example, A software has to be developed and a team is divided to work on a feature of the product and is allowed to work as they want. One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.

SDLC Cycle

SDLC Cycle represents the process of developing software. Below is the diagrammatic representation of the SDLC cycle:

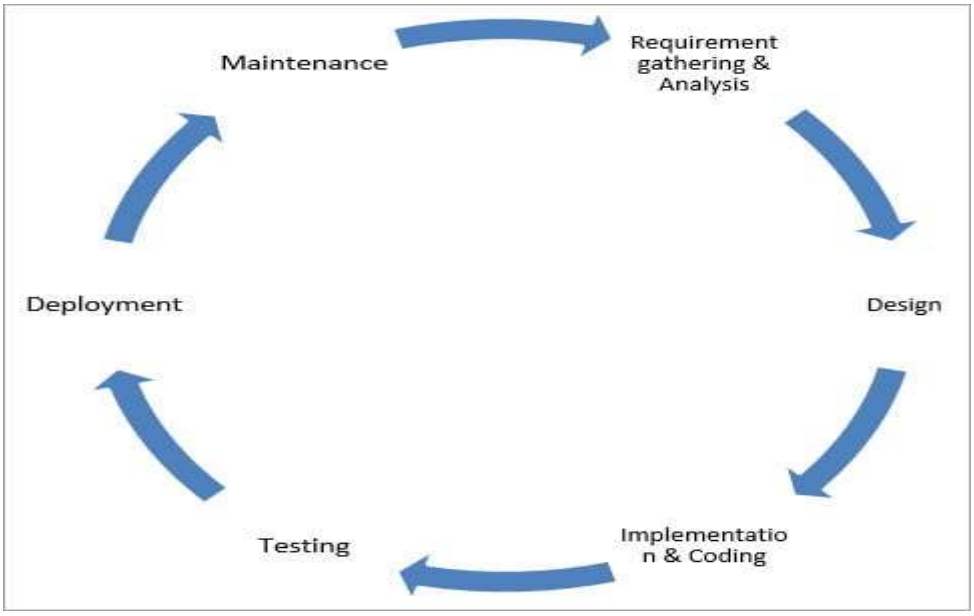


Figure 1.2: SDLC Cycle

SDLC Phases

Given below are the various phases:

a. Requirement gathering and analysis
b. Design
c. Implementation or coding
d. Testing
e. Deployment
f. Maintenance

a) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

For Example, A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

b) Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

c) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

d) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed. Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

e) Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

f) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

1.9 Validation and Verification

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:

Verification: *Are we building the product right?*
Validation: *Are we building the right product?*

a) **Verification:**

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.

Verification is **Static Testing**.

Activities involved in verification:

i. Inspections
ii. Reviews
iii. Walkthroughs
iv. Desk-checking

b) **Validation:**

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.

Validation is the **Dynamic Testing**.

Activities involved in validation:

i. Black box testing

ii. White box testing

iii. Unit testing

iv. Integration testing

1.10 Characteristics of Good Software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational: This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional: This aspect is important when the software is moved from one platform to another:

- Portability

- Interoperability
- Reusability
- Adaptability

Maintenance: This aspect briefs about how well software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

1.11 Software Application:-

S/W applications can be grouped into 8 different areas as given below.

i) System software

ii)Real-time software

iii) Embeded software

iv) Business software

v) PC software

vi) AI software

vii) Web-based s/w

viii)Engineering & scientific software

SELF ASSESSMENT TEST

1. What is software engineering?
2. Difference between Software Engineering and Computer Engineering.
3. Explain the characteristics of good software.
4. Difference between the program and software.
5. What is software crisis?
6. Explain the types of software in detail.
7. Differentiate between Verification and Validation.
8. Define the terms Program ,Software, Software Engineering.
9. Describe the Software Development Life Cycle. Explain the importance of SDLC in software engineering.
10. Define the terms Software process, Software project, Software Product. How these are related to each other?

SUBJECT: Software Engineering	
COURSE CODE: MCA 14	AUTHOR:
LESSON NO. 2	
Software Development Life Cycle Models	

Structure

2.0 Learning Objective

This lesson emphasis on need of software development life cycle and describe the various phases of SDLC . Further it also discuss various types of SDLC Models with their advantages and disadvantages to each other.

2.1 Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

2.2 The Need for a Software Life Cycle Model

The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated by using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure.

2.3 SDLC Cycle

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:

The stages of SDLC are as follows:

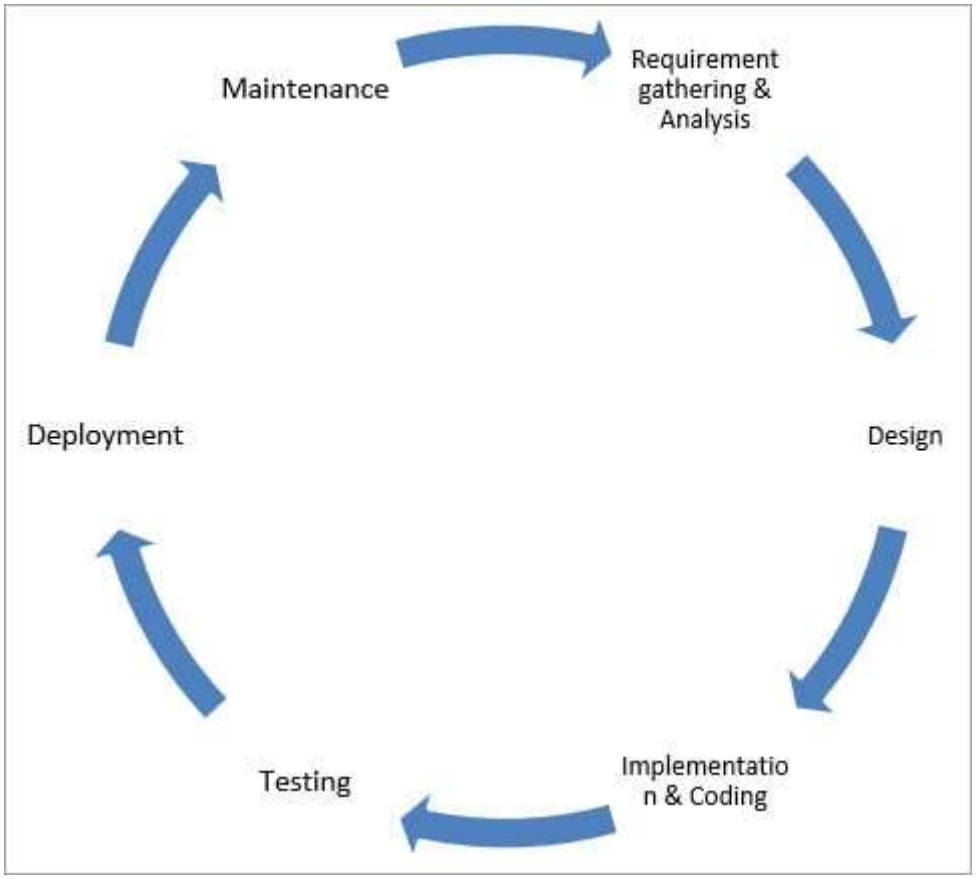


Figure 2.1 Software Development Life Cycle Model

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

Stage1: Requirement gathering and analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage2: Design

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Coding

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

2.4 Software Development Life Cycle Models

A software life cycle model defines entry and exit criteria for every phase. A phase can start only if its phase-entry criteria have been satisfied. So without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle models it becomes difficult for software project managers to monitor the progress of the project. Different software life cycle models been proposed by researchers. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

2.4.1 Classical Waterfall Model

2.4.2 Iterative Waterfall Model

2.4.3 Prototyping Model

2.4.4 Evolutionary Model

2.4.5 Spiral Model

2.4.1 Classical Waterfall Model

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it cannot be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model. Classical waterfall model divides the life cycle into the following phases as shown in fig.2.1:

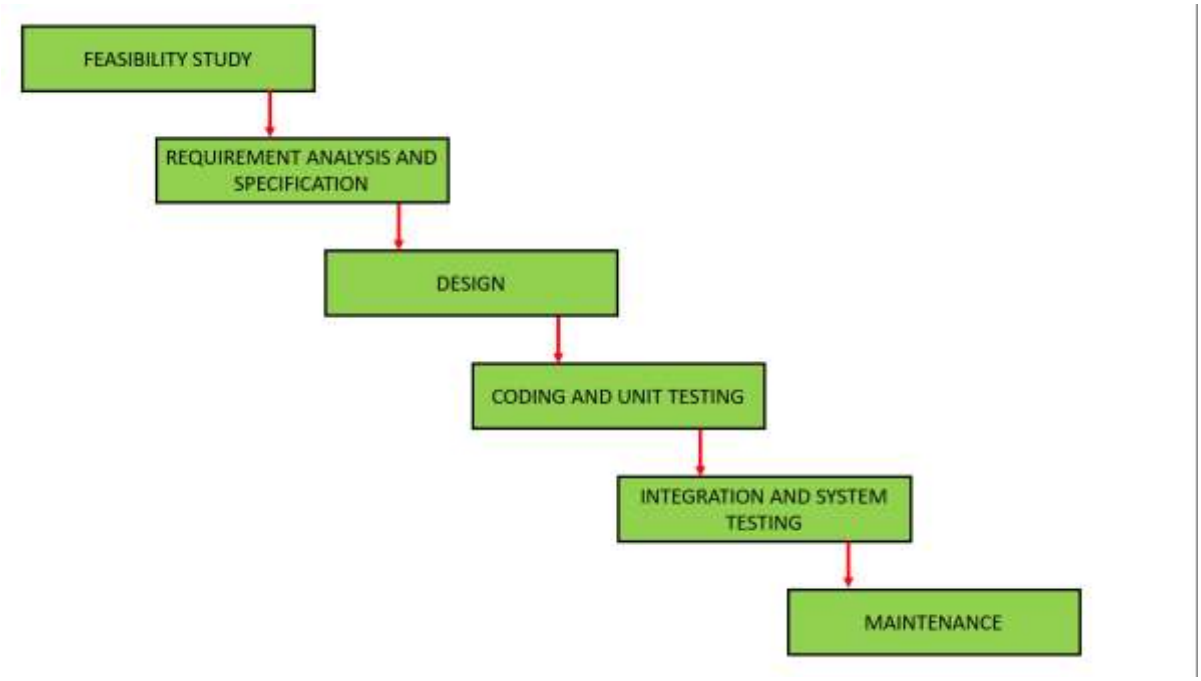


Fig **2.2: Classical Waterfall Model**

Feasibility study - The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the system.

- After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.

- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development. Requirements analysis and specification: - The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

Requirements gathering and analysis

- Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed. The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document. The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are functional requirements, the non-functional requirements, and the goals of implementation.

Design: - The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

- Traditional design approach -Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.

- Object-oriented design approach -In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design. Coding and unit testing:-The purpose of the coding phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested. During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage. Integration and system testing: -Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α – testing: It is the system testing performed by the development team.

- β –testing: It is the system testing performed by a friendly set of customers.

- Acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product. System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

Maintenance: -Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratios. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.

- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.

- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

Advantages and Disadvantages of Water fall Model

Advantages

- Simple to use and understand
- Management simplicity thanks to its rigidity: every phase has a defined result and process review
- Development stages go one by one
- Perfect for the small or mid-sized projects where requirements are clear and not equivocal
- Easy to determine the key points in the development cycle
- Easy to classify and prioritize tasks

Disadvantages

- The software is ready only after the last stage is over
- High risks and uncertainty
- Not the best choice for complex and object-oriented projects
- Inappropriate for the long-term projects
- Integration is done at the very end, which does not give the option of identifying the problem in advance
- The progress of the stage is hard to measure while it is still in the development

Shortcomings of the classical waterfall model

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

2.4.2 Iterative Waterfall Model

To overcome the major shortcomings of the classical waterfall model, we come up with the iterative waterfall model as shown in figure 2.2.

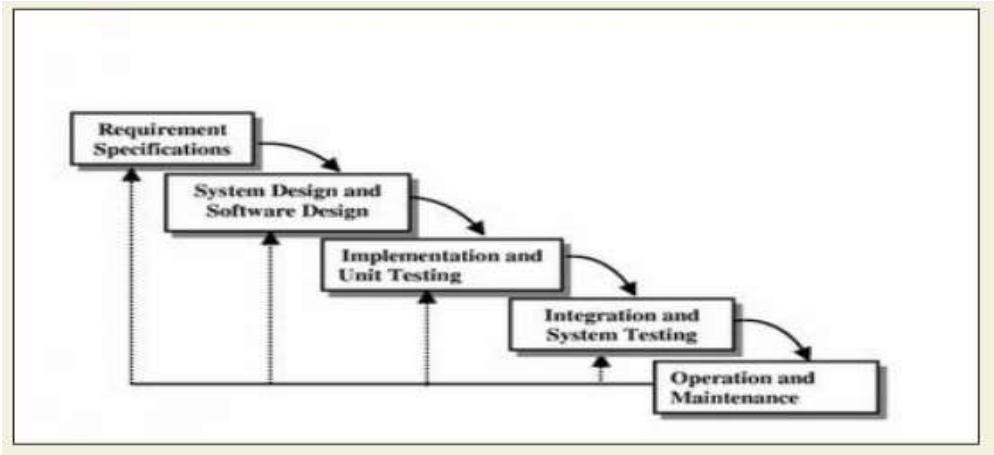


Fig 2.3: Iterative Waterfall Model

Here, we provide feedback paths for error correction as & when detected later in a phase. Though errors are inevitable, but it is desirable to detect them in the same phase in which they occur. If so, this can reduce the effort to correct the bug. The advantage of this model is that there is a working model of the system at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget. The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

Advantages and Disadvantages of Iterative Waterfall Model

Advantages

- Some functions can be quickly developed at the beginning of the development lifecycle
- The paralleled development can be applied
- The progress is easy measurable
- The shorter iteration is - the easier testing and debugging stages are
- It is easier to control the risks as high-risk tasks are completed first
- Problems and risks defined within one iteration can be prevented in the next sprints
- Flexibility and readiness to the changes in the requirements
- Iterative model requires more resources than the waterfall model

Disadvantages

- Constant management is required
- Issues with architecture or design may occur because not all the requirements are foreseen during the short planning stage
- Bad choice for the small projects
- The process is difficult to manage
- The risks may not be completely determined even at the final stage of the project
- Risks analysis requires involvement of the highly-qualified specialists

2.4.3 Prototyping Model

A prototype is a toy implementation of the system as shown in figure 2.3. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system. Need for a prototype in software development. There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- How the screens might look like
- How the user interface would behave
- How the system would produce outputs.

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product. A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues. A prototype of the actual product is preferred in situations such as:

- User requirements are not complete
- Technical issues are not clear

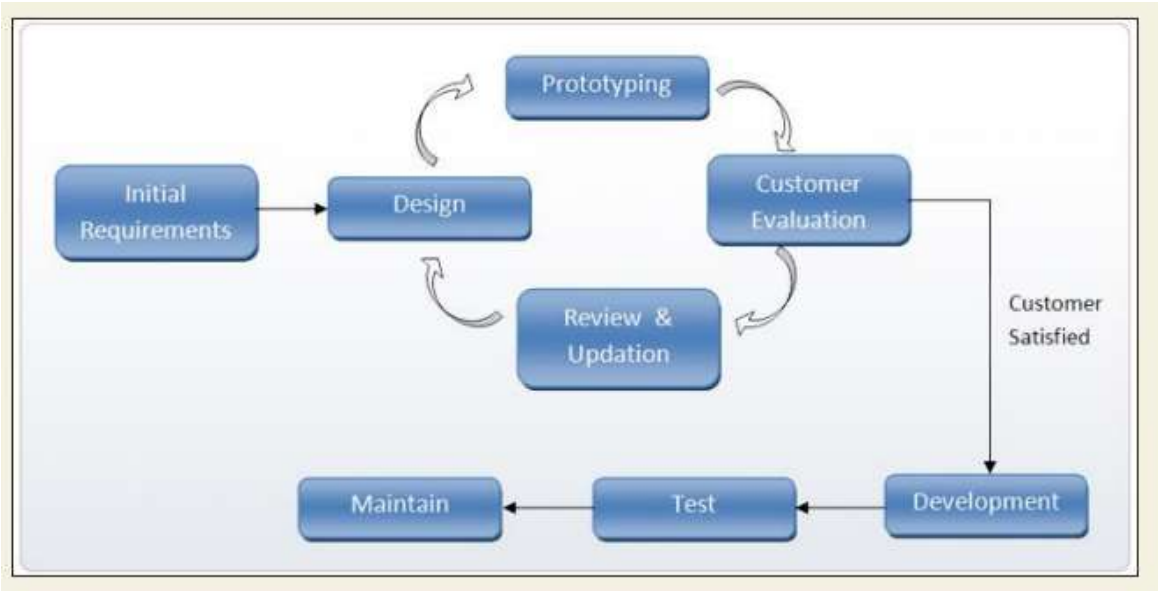


Fig 2.4: Prototyping Model

Advantages and Disadvantages of Prototyping Model

Advantages

- Users are actively involved in the development.
- It provides a better to users, as users have natural tendency to change their mind in specifying requirements and this method of developing systems supports the user tendency.
- Since in this methodology a working model of the system is provided , the users get a better understanding of the system being developed.
- Errors can be detected much earlier in the system is mode side by side.
- Quicker user feedback is available leading to better solutions.

Disadvantages

- Leads to implementing and then repairing way of building systems.
- Practically , the methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

2.4.4 Evolutionary Model

It is also called successive versions model or incremental model. Evolutionary process model resembles the iterative enhancement model. The same phases are defined for the waterfall model occurs here in a cyclical fashion. This model differs from the iterative enhancement model in the sense that this does not require a useful product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

At first, a simple working model is built. Subsequently it undergoes functional improvements & we keep on adding new functions till the desired system is built. Applications:

- Large projects where you can easily find modules for incremental implementation. Often used when the customer wants to start using the core features rather than waiting for the full software.

- Also used in object oriented software development because the system can be easily portioned into units in terms of objects.

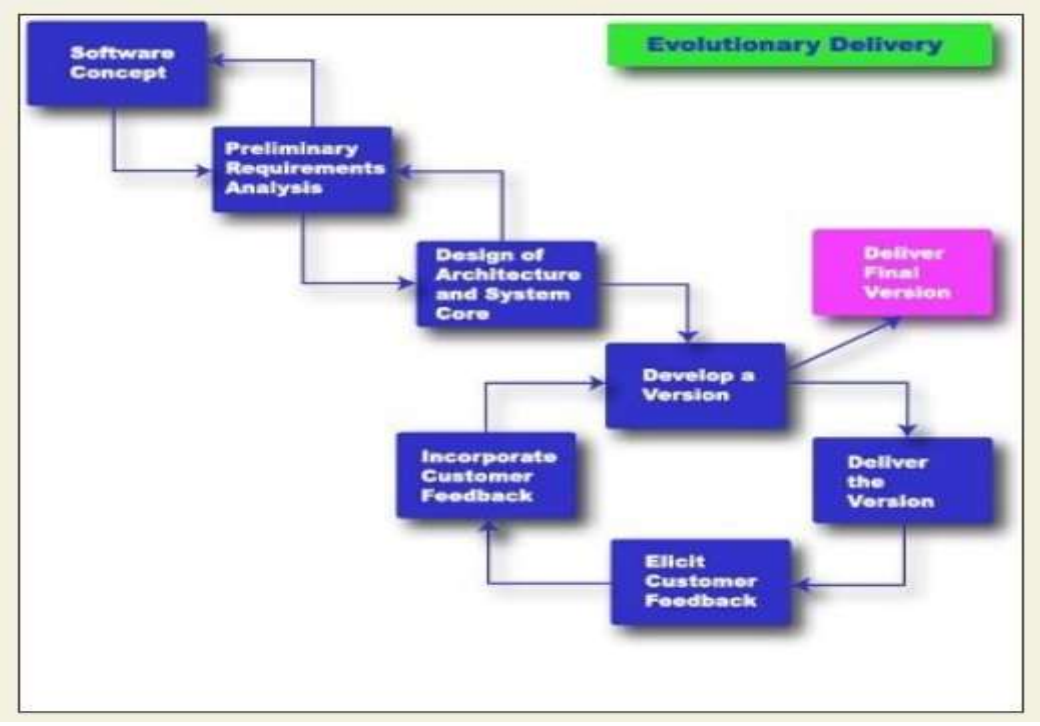


Fig 2.5: Evolutionary Model

Advantages and Disadvantages of Evolutionary Model

Advantages:

- User gets a chance to experiment partially developed system
- Reduce the error because the core modules get tested thoroughly.

Disadvantages:

- It is difficult to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented & delivered.

2.4.5 Spiral Model

Spiral Model is a risk-driven software development process model. It is a combination of waterfall model and iterative model. Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process.

The Spiral model of software development is shown in fig. 2.6.

Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress. The spiral model in software engineering was first mentioned by Barry Boehm in his 1986 paper.

The development process in Spiral model in SDLC, starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase. The below figure very well explain Spiral Model:

The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study, the next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. 4.1. The following activities are carried out during each phase of a spiral model.

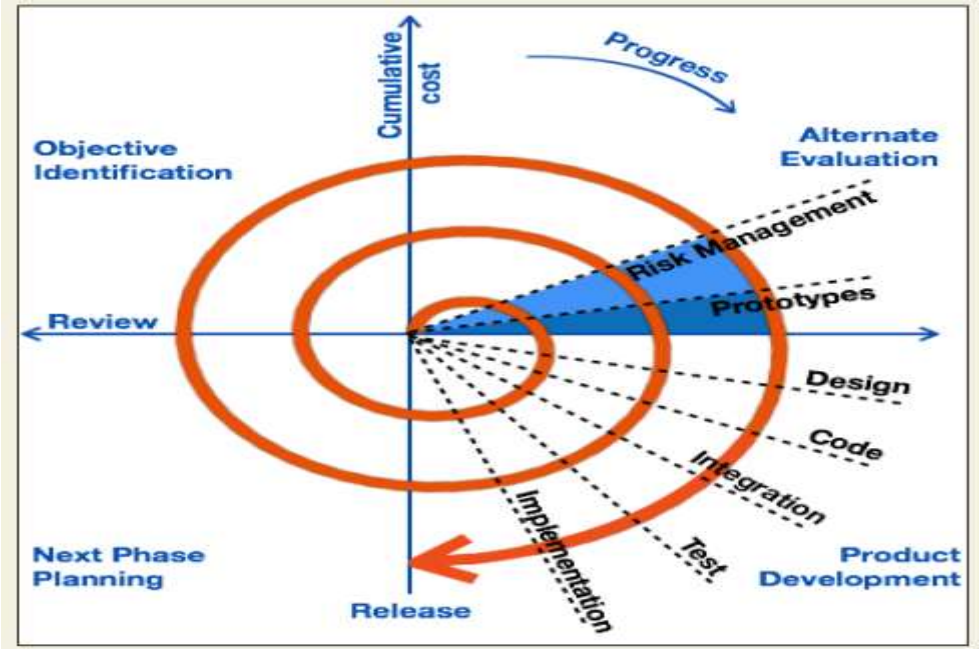


Fig 2.6: Spiral Model

First quadrant (Objective Setting)

• During the first quadrant, it is needed to identify the objectives of the phase. • Examine the risks associated with these objectives.

Second Quadrant (Risk Assessment and Reduction)

• A detailed analysis is carried out for each identified project risk. • Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

Third Quadrant (Development and Validation)

• Develop and validate the next level of the product after resolving the identified risks.

Fourth Quadrant (Review and Planning)

• Review the results achieved so far with the customer and plan the next iteration around the spiral.

• Progressively more complete version of the software gets built with each iteration around the spiral.

Advantages and Disadvantages of Spiral Model

Advantages

- High amount of risk analysis
- Good for large and mission-critical projects
- Software is produced early in the software life cycle.

Disadvantages

- Can be costly model to use
- Risk analysis requires highly specific expertise.
- Project's success is highly depedenet on the risk analysis phase.

Circumstances to use spiral model

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

2.5 Comparison of different life-cycle models

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model cannot be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases. This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks. The prototyping

model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer. The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects. The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer resentment. On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches. Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new system. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront capital outlay. The customer can order the incremental versions as and when he can afford them.

SELF ASSESSMENT TEST

1. What is software development life cycle model?
2. Why we need software development models.
3. What is waterfall model?
4. Explain prototype model.
5. Difference between the waterfall and spiral model.

6. Explain in detail spiral model. How risk is covered in this model?

****

SUBJECT: Software Engineering

COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 3	

Software Project Managenment

3.1 Introduction

3.1.1. Software Project

3.1.2. Software Project Manager

3.1.3. Software Management Activities

3.2 Project Estimation Techniques

3.2.1 Decomposition Technique

3.2.2 Empirical Estimation Technique

3.3 Project Scheduling

3.3.1 Resource management

3.3.2 Project Risk Management

3.3.3 Project Execution & Monitoring

3.3.4 Project Communication Management

3.4 Configuration Management

3.4.1 Baseline

3.4.2 Change Control

3.4.3 Project Management Tools

3.1 INTRODUCTION

The job pattern of an IT company engaged in software development can be seen split in two parts:

- Software Creation
- Software Project Management

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

- Every project may have a unique and distinct goal.

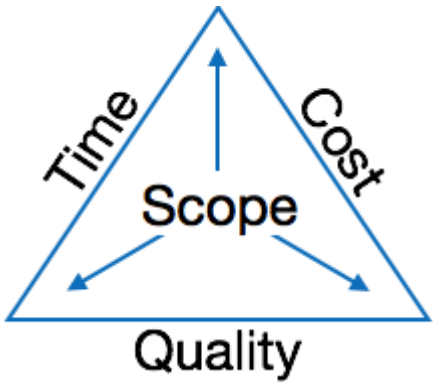
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

3.1.1 Software Project

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management

Software is said to be an intangible product. Software development is a kind of all new streams in world business and there's very little experience in building software products. Most software products are tailor made to fit client's requirements. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.



The image above shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver

the project as per scheduled. There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factors can severely impact the other two.

Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

3.1.2 Software Project Manager

A software project manager is a person who undertakes the responsibility of executing the software project. Software project manager is thoroughly aware of all the phases of SDLC that the software would go through. Project manager may never directly involves in producing the end product but he controls and manages the activities involved in production.

A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, and quality and customer satisfaction.

Let us see few responsibilities that a project manager shoulders -

Managing People

- Act as project leader
- Liaison with stakeholders
- Managing human resources
- Setting up reporting hierarchy etc.

Managing Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems
- Act as project spokesperson

3.1.3 Software Management Activities

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

- **Project Planning**
- **Scope Management**
- **Project Estimation**

Project Planning

Software project planning is task, which is performed before the production of software actually starts. It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production. Project planning may include the following:

Scope Management

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

During Project Scope management, it is necessary to -

- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

  Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. An effort required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware

- Additional software or tools, licenses etc.
- Skilled personnel with task-specific skills
- Travel involved
- Communication
- Training and support

3.2 Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques-

3.2.1 Decomposition Technique

This technique assumes the software as a product of various compositions.

There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

3.2.2 Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation.These formulae are based on LOC or FPs.

- **Putnam Model**

  This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

- **COCOMO**

  COCOMO stands for COnstructive COst MOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

3.3 Project Scheduling

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks and project milestones and arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to -

- Break down the project tasks into smaller, manageable form
- Find out various tasks and correlate them
- Estimate time frame required for each task
- Divide time into work-units
- Assign adequate number of work-units for each task
- Calculate total time required for the project from start to finish

3.3.1 Resource management

All elements used to develop a software product may be assumed as resource for that project. This may include human resource, productive tools and software libraries.

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers the development of project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes -

- Defining proper organization project by creating a project team and allocating responsibilities to each team member
- Determining resources required at a particular stage and their availability
- Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.

3.3.2 Project Risk Management

Risk management involves all activities pertaining to identification, analysing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Risk Management Process

There are following activities involved in risk management process:

- **Identification -** Make note of all possible risks, which may occur in the project.
- **Categorize -** Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.
- **Manage -** Analyses the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.
- **Monitor -** Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.

3.3.3 Project Execution & Monitoring

In this phase, the tasks described in project plans are executed according to their schedules.

Execution needs monitoring in order to check whether everything is going according to the plan. Monitoring is observing to check the probability of risk and taking measures to address the risk or report the status of various tasks.

These measures include -

- **Activity Monitoring -** All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.

- **Status Reports -** The reports contain status of activities and tasks completed within a given time frame, generally a week. Status can be marked as finished, pending or work-in-progress etc.
- **Milestones Checklist -** Every project is divided into multiple phases where major tasks are performed (milestones) based on the phases of SDLC. This milestone checklist is prepared once every few weeks and reports the status of milestones.

3.3.4 Project Communication Management

Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers.

Communication can be oral or written. Communication management process may have the following steps:

- **Planning** - This step includes the identifications of all the stakeholders in the project and the mode of communication among them. It also considers if any additional communication facilities are required.
- **Sharing** - After determining various aspects of planning, manager focuses on sharing correct information with the correct person on correct time. This keeps everyone involved the project up to date with project progress and its status.
- **Feedback** - Project managers use various measures and feedback mechanism and create status and performance reports. This mechanism ensures that input from various stakeholders is coming to the project manager as their feedback.
- **Closure** - At the end of each major event, end of a phase of SDLC or end of the project itself, administrative closure is formally announced to update every stakeholder by sending email, by distributing a hardcopy of document or by other mean of effective communication.

After closure, the team moves to next phase or project.

3.4 Configuration Management

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as "the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items".

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with prior approval of higher management, as there is a possibility of cost and time overrun.

3.4.1 Baseline

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase, is baselined. CM keeps check on any changes done in software.

3.4.2 Change Control

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps -

- **Identification** - A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation** - Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis** - The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control** - If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.

- **Execution** - If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.
- **Close request** - The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

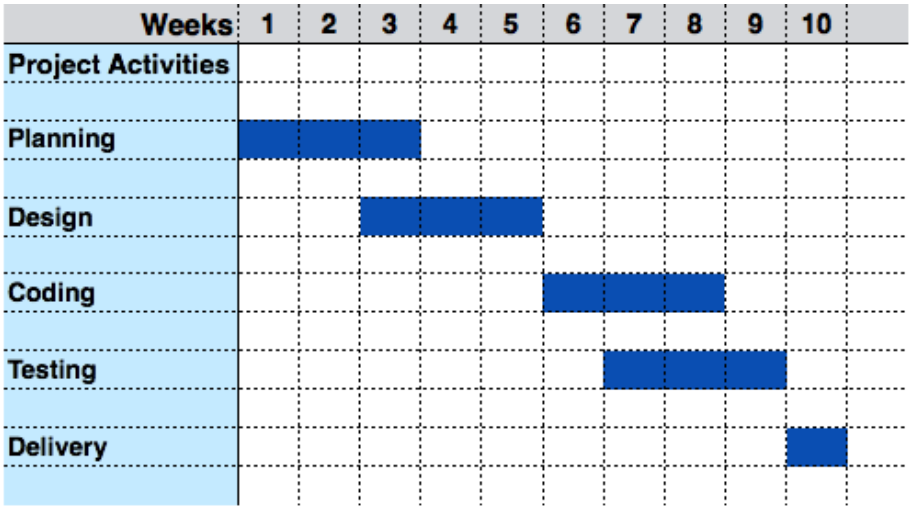
3.4.3 Project Management Tools

The risk and uncertainty rises multifold with respect to the size of the project, even when the project is developed according to set methodologies.

There are tools available, which aid for effective project management. A few are described -

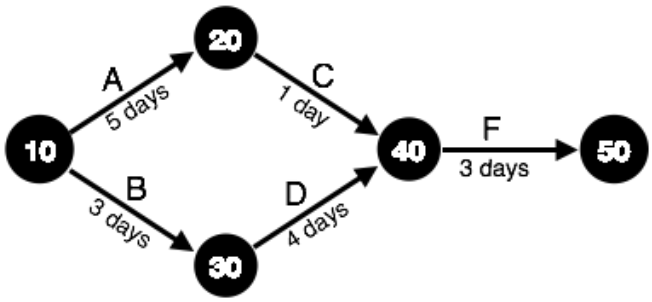
Gantt chart

A Gantt chart was devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.



PERT Chart

PERT (Program Evaluation & Review Technique) chart is a tool that depicts project as network diagram. It is capable of graphically representing main events of project in both parallel and consecutive way. Events, which occur one after another, show dependency of the later event over the previous one.

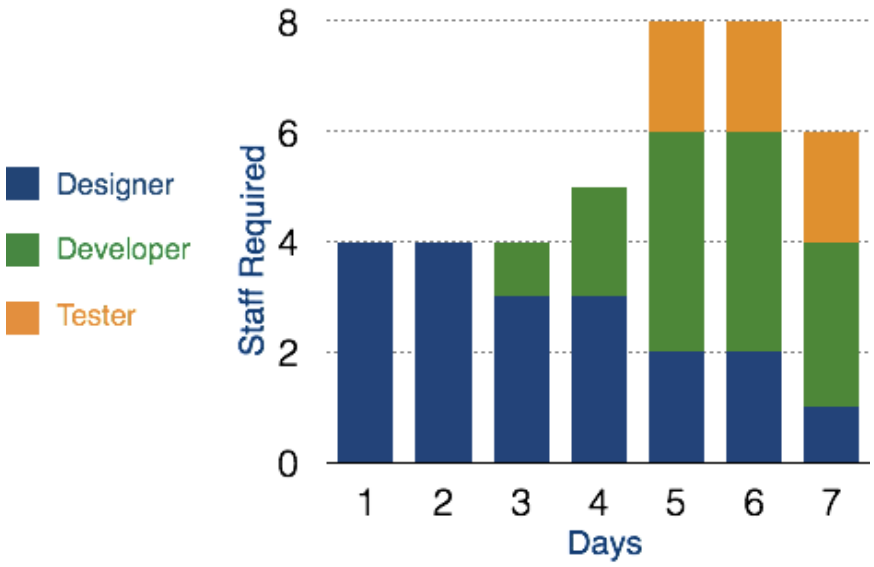


Events are shown as numbered nodes. They are connected by labeled arrows depicting sequence of tasks in the project.

Resource Histogram

This is a graphical tool that contains bar or chart representing number of resources (usually skilled staff) required over time for a project event (or phase). Resource Histogram is an effective tool for staff planning and coordination.

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



3.4.4 Critical Path Analysis

This tool is useful in recognizing interdependent tasks in the project. It also helps to find out the shortest path or critical path to complete the project successfully. Like PERT diagram, each event is allotted a specific time frame. This tool shows dependency of event assuming an event can proceed to next only if the previous one is completed.

The events are arranged according to their earliest possible start time. Path between start and end node is critical path which cannot be further reduced and all events require to be executed in same order.

3.5 Project size estimation techniques

Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

3.5.1 Lines of Code (LOC): As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non comment lines of code
- KDSI- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.

- Simple to use.

Disadvantages:

- A different programming language contains different number of lines.
- No proper industry standard exists for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

3.5.2 Number of entities in ER diagram: ER model provides a static view of the project. It describes the entities and its relationships. The number of entities in ER model can be used to measure the estimation of size of project. Number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

Advantages:

- Size estimation can be done during initial stages of planning.
- Number of entities is independent of programming technologies used.

Disadvantages:

- No fixed standards exist. Some entities contribute more project size than others.
- Just like FPA, it is less used in cost estimation model. Hence, it must be converted to LOC.

3.5.3 Total number of processes in detailed data flow diagram: Data Flow Diagram (DFD) represents the functional view of software. The model depicts the main processes/functions involved in software and flow of data between them. Utilization of number of functions in DFD to predicts software size. Already existing processes of similar type are studied and used to estimate the size of the process. Sum of the estimated size of each process gives the final estimated size.

Advantages:

- It is independent of programming language.
- Each major process can be decomposed into smaller processes. This will increase the accuracy of estimation

Disadvantages:

- Studying similar kind of processes to estimate size takes additional time and effort.
- All software projects are not required to construction of DFD.

3.5.4 Function Point Analysis: In this method, the number and type of functions supported by the software are utilized to find FPC (function point count). The steps in function point analysis are:

- Count the number of functions of each proposed type.
- Compute the Unadjusted Function Points (UFP).
- Find Total Degree of Influence (TDI).
- Compute Value Adjustment Factor (VAF).
- Find the Function Point Count (FPC).

The explanation of above points given below:

- **Count the number of functions of each proposed type:** Find the number of functions belonging to the following types:
    - External Inputs: Functions related to data entering the system.
    - External outputs: Functions related to data exiting the system.
    - External Inquiries: They lead to data retrieval from system but don't change the system.
    - Internal Files: Logical files maintained within the system. Log files are not included here.
    - External interface Files: These are logical files for other applications which are used by our system.
- **Compute the Unadjusted Function Points (UFP):** Categorise each of the five function types as simple, average or complex based on their complexity. Multiply count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

- **Find Total Degree of Influence:** Use the '14 general characteristics' of a system to find the degree of influence of each of them. The sum of all 14 degrees of influences will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change. Each of above characteristics is evaluated on a scale of 0-5.
- **Compute Value Adjustment Factor(VAF):** Use the following formula to calculate VAF VAF = (TDI * 0.01) + 0.65
- **Find the Function Point Count:** Use the following formula to calculate FPC FPC = UFP * VAF

Advantages:

- It can be easily used in the early stages of project planning.
- It is independent on the programming language.
- It can be used to compare different projects even if they use different technologies (database, language etc.).

Disadvantages:

- It is not good for real time systems and embedded systems.
- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

3.6 COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

E_i=a*(KDLOC)b

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

3.5.1 Organic

3.5.2 Semidetached

3.5.3 Embedded

3.5.1 Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of**

this type of projects are simple business systems, simple inventory management systems, and data processing systems.

3.5.2 Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3.5.3 Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

- Basic Model
- Intermediate Model
- Detailed Model

Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

Effort=a_1*(KLOC) a_2 PM

Tdev=b_1*(efforts)b_2 Months

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1,a_2,b_1,b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

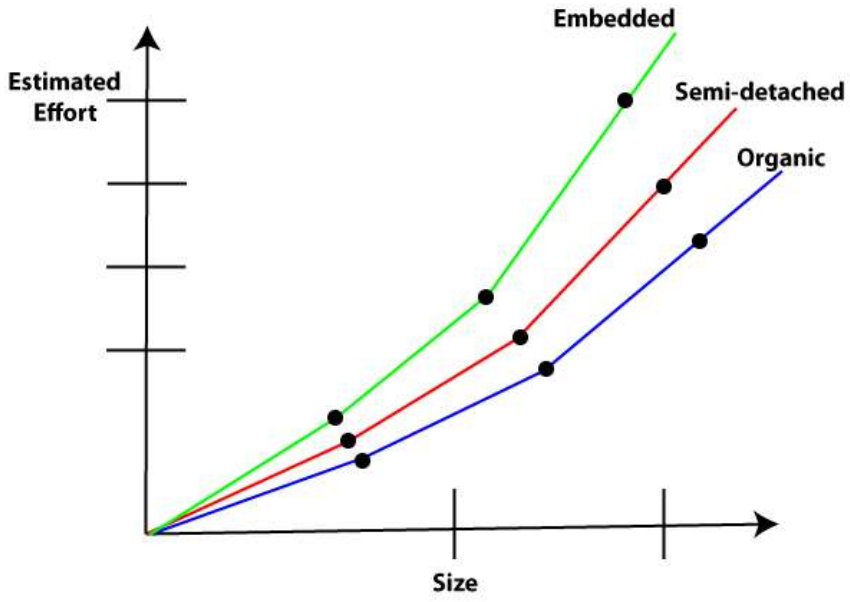
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

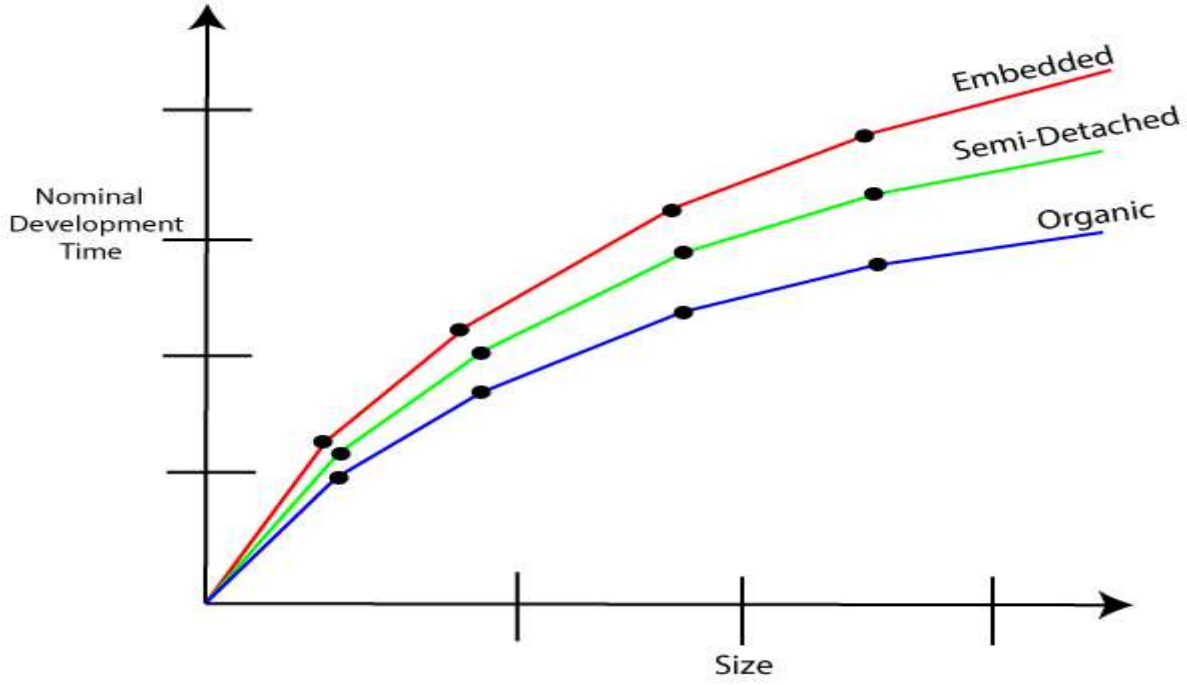
Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a_1*(KLOC) a_2 PM

Tdev=b_1*(efforts)b_2 Months

Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM

D = 2.5 * (1295.31)0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400)1.12=2462.79 PM

D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM

D = 2.5 * (4772.8)0.32 = 38 PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM

D=2.5(1133.12)0.35=29.3PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

Intermediate Model: The basic COCOMO model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability

- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

Intermediate COCOMO equation:

$$E=a_i (KLOC)\ b_i * EAF$$

$$D=c_i (E)d_i$$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver's effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed COCOMO, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

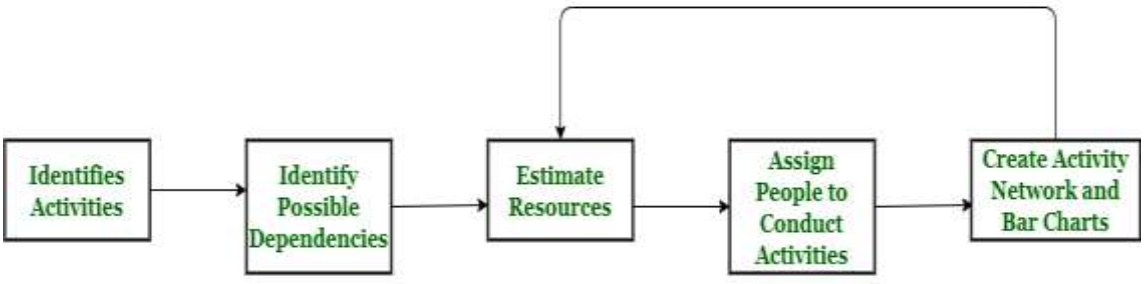
1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

3.6 Project Scheduling

A schedule in your project's time table actually consists of sequenced activities and milestones that are needed to be delivered under a given period of time.

Project schedule simply means a mechanism that is used to communicate and know about that tasks are needed and has to be done or performed and which organizational resources will be given or allocated to these tasks and in what time duration or time frame work is needed to be performed. Effective project scheduling leads to success of project, reduced cost, and increased customer satisfaction. Scheduling in project management means to list out activities, deliverables, and milestones within a project that are delivered. It contains more notes than your average weekly planner notes. The most common and important form of project schedule is Gantt chart.



Project Scheduling Process

Process:

The manager needs to estimate time and resources of project while scheduling project. All activities in project must be arranged in a coherent sequence that means activities should be arranged in a logical and well-organized manner for easy to understand. Initial estimates of project can be made optimistically which means estimates can be made when all favorable things will happen and no threats or problems take place.

The total work is separated or divided into various small activities or tasks during project schedule. Then, Project manager will decide time required for each activity or task to get completed. Even some activities are conducted and performed in parallel for efficient performance. The project manager should be aware of fact that each stage of project is not problem-free.

Problems arise during Project Development Stage:

- People may leave or remain absent during particular stage of development.
- Hardware may get failed while performing.
- Software resource that is required may not be available at present, etc.

The project schedule is represented as set of chart in which work-breakdown structure and dependencies within various activities are represented. To accomplish and complete project within a given schedule, required resources must be available when they are needed. Therefore, resource estimation should be done before starting development.

Resources required for Development of Project:

- Human effort
- Sufficient disk space on server
- Specialized hardware
- Software technology
- Travel allowance required by project staff, etc.

Advantages of Project Scheduling:

There are several advantages provided by project schedule in our project management:

- It simply ensures that everyone remains on same page as far as tasks get completed, dependencies, and deadlines.
- It helps in identifying issues early and concerns such as lack or unavailability of resources.
- It also helps to identify relationships and to monitor process.
- It provides effective budget management and risk mitigation.

3.7 Personnel Planning

Personnel Planning deals with staffing. Staffing deals with the appoint personnel for the position that is identified by the organizational structure.

It involves:

- Defining requirement for personnel
- Recruiting (identifying, interviewing, and selecting candidates)
- Compensating
- Developing and promoting agent

For personnel planning and scheduling, it is helpful to have efforts and schedule size for the subsystems and necessary component in the system.

At planning time, when the system method has not been completed, the planner can only think to know about the large subsystems in the system and possibly the major modules in these subsystems.

Once the project plan is estimated, and the effort and schedule of various phases and functions are known, staff requirements can be achieved.

From the cost and overall duration of the projects, the average staff size for the projects can be determined by dividing the total efforts (in person-months) by the whole project duration (in months).

Typically the staffs required for the project is small during requirement and design, the maximum during implementation and testing, and drops again during the last stage of integration and testing.

Using the COCOMO model, average staff requirement for various phases can be calculated as the effort and schedule for each method are known.

When the schedule and average staff level for every action are well-known, the overall personnel allocation for the project can be planned.

This plan will indicate how many people will be required for different activities at different times for the duration of the project.

The total effort for each month and the total effort for each step can easily be calculated from this plan.

3.8 Team Structure

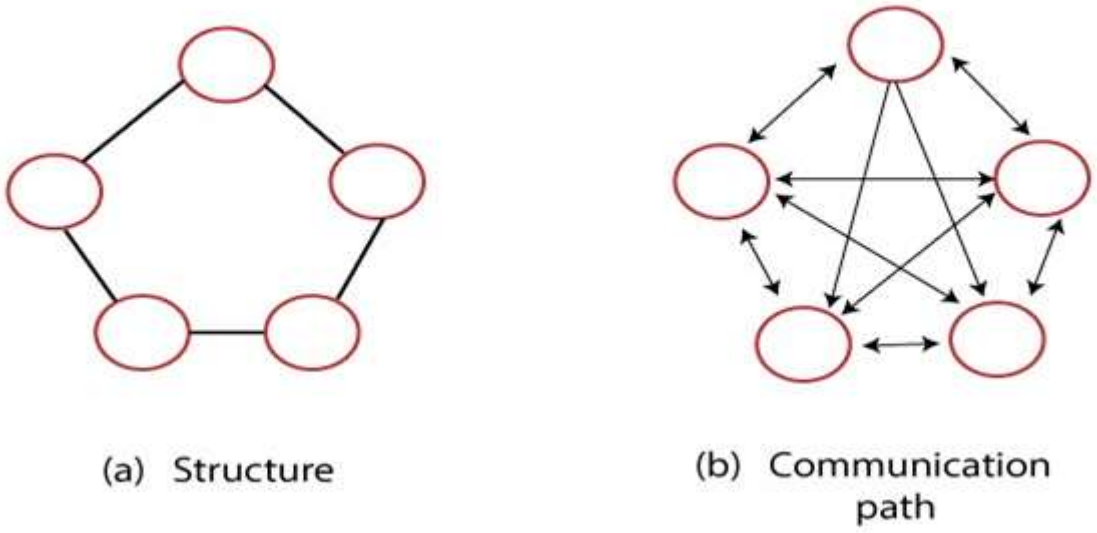
Team structure addresses the issue of arrangement of the individual project teams. There are some possible methods in which the different project teams can be organized. There are primarily three formal team structures: **chief programmer, Ego-less or democratic, and the mixed team organizations** even several other variations to these structures are possible. Problems of various complexities and sizes often need different team structures for the chief solution.

Ego-Less or Democratic Teams

Ego-Less teams subsist of a team of fewer programmers. The objective of the group is set by consensus, and input from each member is taken for significant decisions. Group leadership revolves among the group members. Due to its nature, egoless teams are consistently known as democratic teams.

The structure allows input from all representatives, which can lead to better decisions in various problems. This suggests that this method is well suited for long-term research-type projects that do not have time constraints.

Ego-Less Programming Team structure and communication paths



(a) Structure

(b) Communication path

Chief Programmer Team

A chief-programmer team, in contrast to the ego-less team, has a hierarchy. It consists of a chief-programmer, who has a backup programmer, a program librarian, and some programmers.

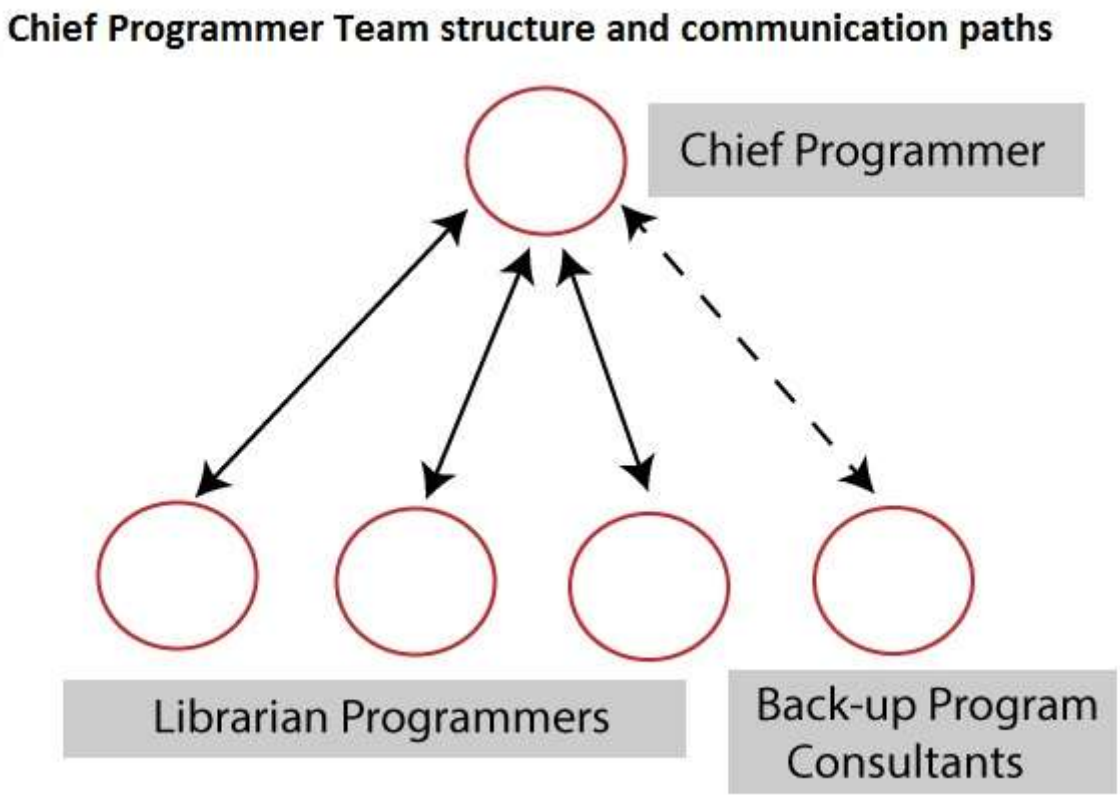
The chief programmer is essential for all major technical decisions of the project.

He does most of the designs, and he assigns coding of the different part of the design to the programmers.

The backup programmer uses the chief programmer makes technical decisions, and takes over the chief programmer if the chief programmer drops sick or leaves.

The program librarian is vital for maintaining the documentation and other communication-related work.

This structure considerably reduces interpersonal communication. The communication paths, as shown in fig:



Controlled Decentralized Team

(Hierarchical Team Structure)

A third team structure known as the controlled decentralized team tries to combine the strength of the democratic and chief programmer teams.

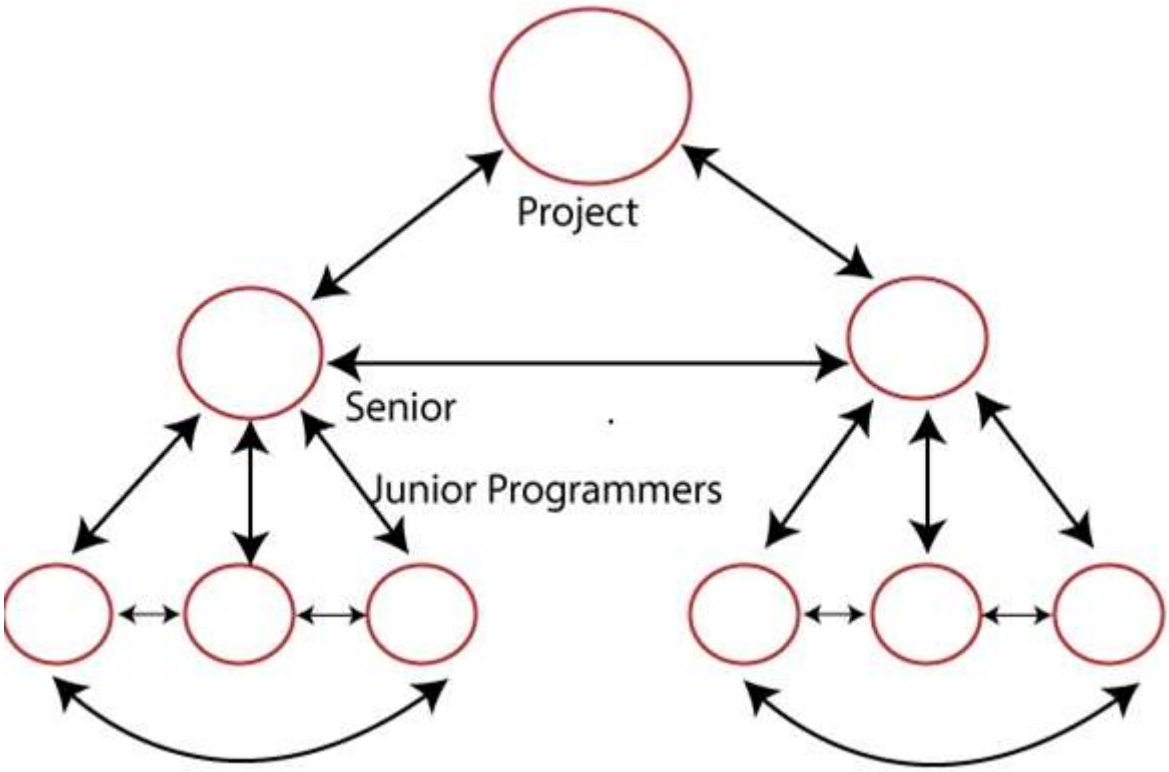
It consists of project leaders who have a class of senior programmers under him, while under every senior programmer is a group of a junior programmer.

The group of a senior programmer and his junior programmers behave like an ego-less team, but communication among different groups occurs only through the senior programmers of the group.

The senior programmer also communicates with the project leader.

Such a team has fewer communication paths than a democratic team but more paths compared to a chief programmer team.

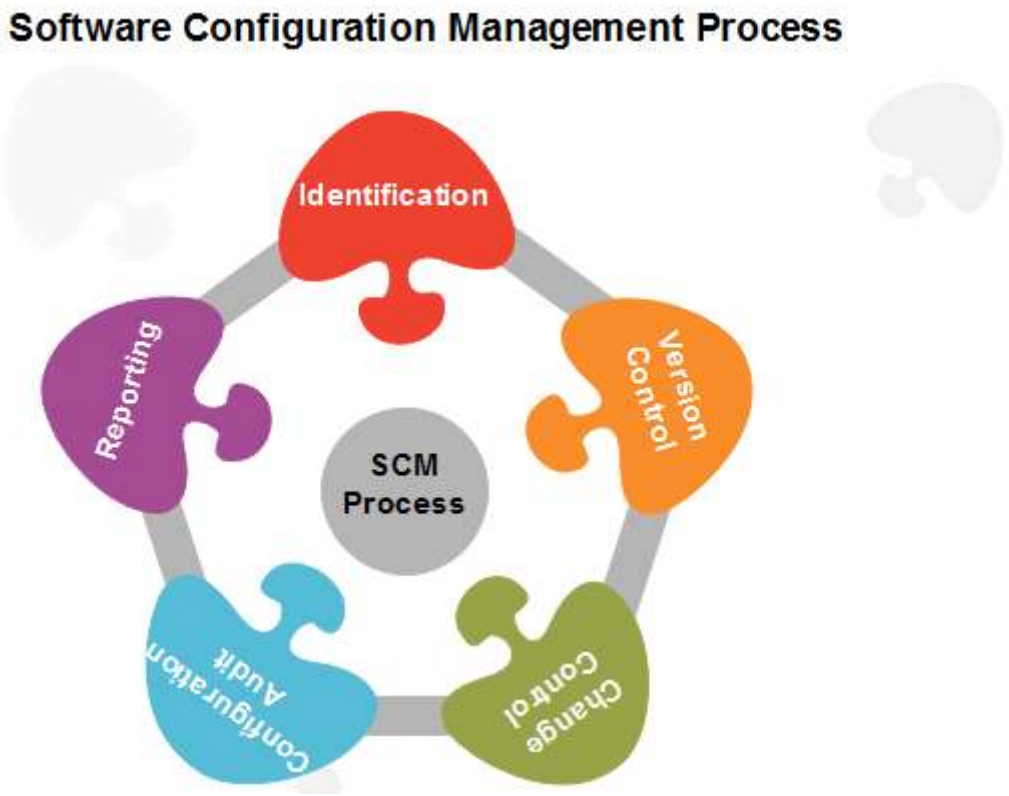
This structure works best for large projects that are reasonably straightforward. It is not well suited for simple projects or research-type projects.



3.9 Software Configuration Management (SCM)

It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

- Identification of objects in the software configuration
- Version Control
- Change Control
- Configuration Audit
- Status Reporting



Identification

Basic Object: Unit of Text created by a software engineer during analysis, design, code, or test.

Aggregate Object: A collection of essential objects and other aggregate objects. Design Specification is an aggregate object.

Each object has a set of distinct characteristics that identify it uniquely: a name, a description, a list of resources, and a "realization."

The interrelationships between configuration objects can be described with a Module Interconnection Language (MIL).

Version Control

Version Control combines procedures and tools to handle different version of configuration objects that are generated during the software process.

Clemm defines version control in the context of SCM: Configuration management allows a user to specify the alternative configuration of the software system through the selection of appropriate

versions. This is supported by associating attributes with each software version, and then allowing a configuration to be specified [and constructed] by describing the set of desired attributes.

Change Control

James Bach describes change control in the context of SCM is: Change Control is Vital. But the forces that make it essential also make it annoying.

We worry about change because a small confusion in the code can create a big failure in the product. But it can also fix a significant failure or enable incredible new capabilities.

We worry about change because a single rogue developer could sink the project, yet brilliant ideas originate in the mind of those rogues, and

A burdensome change control process could effectively discourage them from doing creative work.

A change request is submitted and calculated to assess technical merit; potential side effects, the overall impact on other configuration objects and system functions, and projected cost of the change.

The results of the evaluations are presented as a change report, which is used by a change control authority (CCA) - a person or a group who makes a final decision on the status and priority of the change.

The "check-in" and "check-out" process implements two necessary elements of change control-**access control** and **synchronization control**.

Access Control governs which software engineers have the authority to access and modify a particular configuration object.

Synchronization Control helps to ensure that parallel changes, performed by two different people, don't overwrite one another.

Configuration Audit

SCM audits to verify that the software product satisfies the baselines requirements and ensures that what is built and what is delivered.

SCM audits also ensure that traceability is maintained between all CIs and that all work requests are associated with one or more CI modification.

SCM audits are the "**watchdogs**" that ensures that the integrity of the project's scope is preserved.

Status Reporting

Configuration Status reporting (sometimes also called status accounting) providing accurate status and current configuration data to developers, testers, end users, customers and stakeholders through admin guides, user guides, FAQs, Release Notes, Installation Guide, Configuration Guide, etc.

3.10 What is Risk?

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

We need to differentiate risks, as potential issues, from the current problems of the project.

Different methods are required to address these two kinds of issues.

For example, staff storage, because we have not been able to select people with the right technical skills is a current problem, but the threat of our technical persons being hired away by the competition is a risk.

Risk Management

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

- Project risks
- Technical risks
- Business risks

Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

- **Risk Management Activities**

Risk management consists of three main activities, as shown in fig:

- **Risk Assessment**

The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:

- The possibility of a risk coming true (denoted as r).
- The consequence of the issues relates to that risk (denoted as s).

Based on these two methods, the priority of each risk can be estimated:

p = r * s

Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

Risk Identification: The project organizer needs to anticipate the risk in the project as early as possible so that the impact of risk can be reduced by making effective risk management planning.

A project can be of use by a large variety of risk. To identify the significant risk, this might affect a project. It is necessary to categories into the different risk of classes.

There are different types of risks which can affect a software project:

1. **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.
4. **Tools risks:** Risks that assume from the software tools and other support software used to create the system.
5. **Requirement risks:** Risks that assume from the changes to the customer requirement and the process of managing the requirements change.
6. **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

Risk Analysis: During the risk analysis process, you have to consider every identified risk and make a perception of the probability and seriousness of that risk.

There is no simple way to do this. You have to rely on your perception and experience of previous projects and the problems that arise in them.

It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

1. The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).
2. The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency), or insignificant.

❋ **Risk Control**

It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods. In fact, most risks need ingenuity on the part of the project manager in tackling the risk.

There are three main methods to plan for risk management:

Avoid the risk: This may take several ways such as discussing with the client to change the requirements to decrease the scope of the work, giving incentives to the engineers to avoid the risk of human resources turnover, etc.

Transfer the risk: This method involves getting the risky element developed by a third party, buying insurance cover, etc.

Risk reduction: This means planning method to include the loss due to risk. For instance, if there is a risk that some key personnel might leave, new recruitment can be planned.

Risk Leverage: To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.

Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.

Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)

Risk planning: The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

For each of the risks, you have to think of the behavior that you may take to minimize the disruption to the plan if the issue identified in the risk occurs.

You also should think about data that you might need to collect while monitoring the plan so that issues can be anticipated.

Again, there is no easy process that can be followed for contingency planning. It rely on the judgment and experience of the project manager.

Risk Monitoring: Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.

QUESTION

Q1. Explain in detail Software Project Management Plan (SPMP)?

Q2. What is COCOMO model? How it is applied on different category of project?

Q3. What is Metric? How it is useful in project size estimation?

Q4. What is Risk? Explain in detail Risk Management Activities.

Q5. Write short note on: Project Scheduling, Personnel Planning, Organization and Team Structures.

****

SUBJECT: Software Engineering	
COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 4	

Software Requirements

4.1 INTRODUCTION

The software requirements are description of features and functionalities of the target system.

Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

4.1.1 Requirement Engineering

The process to gather the software requirements from client analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

4.1.2 Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for the management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by the system analyst after the requirements are collected from various stakeholders. SRS defines how the intended software will interact with hardware, external interfaces,

speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud.

Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the following diagram:

Fig 4.1: Requirement elicitation process

- Requirements gathering - The developers discuss with the client and the end users and know their expectations from the software.
- Organizing Requirements - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- Negotiation & discussion - If the requirements are ambiguous or there are some conflicts in the requirements of various stakeholders, it is then negotiated and discussed with the stakeholders. Requirements may then be prioritized and reasonably compromised. The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.
- Documentation - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements:

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to be gathered, is decided in advance, they follow the pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.

- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visits the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow 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.

4.2 SOFTWARE REQUIREMENT CHARACTERISTICS

Gathering software requirements is the foundation of the entire software development project.

Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements: Requirements, which are related to functional aspect of software fall into this category. They define functions and functionality within and from the software system.

EXAMPLES -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

4.3 Non-Functional Requirements: Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- Must Have: Software cannot be said operational without them.
- Should have: Enhancing the functionality of software.
- Could have: Software can still properly function with these requirements.
- Wish list: These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

4.4 User Interface (UI) requirements

UI is an important part of any software or hardware or hybrid system. Software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system cannot be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategically use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts

during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software
- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement
- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements.

4.5 Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software

Product Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

Let us see some software metrics:

- Size Metrics - LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

Function Point Count is measure of the functionality provided by the software. Function

Point count defines the size of functional aspect of software.

- Complexity Metrics - McCabe's Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.
- Quality Metrics - Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- Process Metrics - In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- Resource Metrics - Effort, time and various resources used, represents metrics for resource measurement.

Questions

Q1. What is Software Requirement Specification (SRS)?

Q2. Difference between functional and non-functional requirements?

Q3. Explain in detail User and interface requirements.

Q4. Explain the role of Software System Analyst?

****

SUBJECT: Software Engineering	
COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 5	

Problem Analysis

5.0 Learning Objective

This lesson emphasis the concept of structured analysis and tools required for structured analysis- DFD, Decision table, decision tree data dictionary, structured charts. Further, this lesson also focus on OOA techniques for OOA –object modelling, dynamic modelling, and functional modelling.

5.1 Structured Analysis

Definition : it is a set of techniques and graphic tools that allows the analyst to understand the system and its activities in a logical way. It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes –

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

Tools of Structured analysis

- Data Flow Diagram(DFD)
- Decision Tables
- Decision Tree
- Data dictionary
- Structured Charts

Fig 5.1 shows logical data description hierarchy

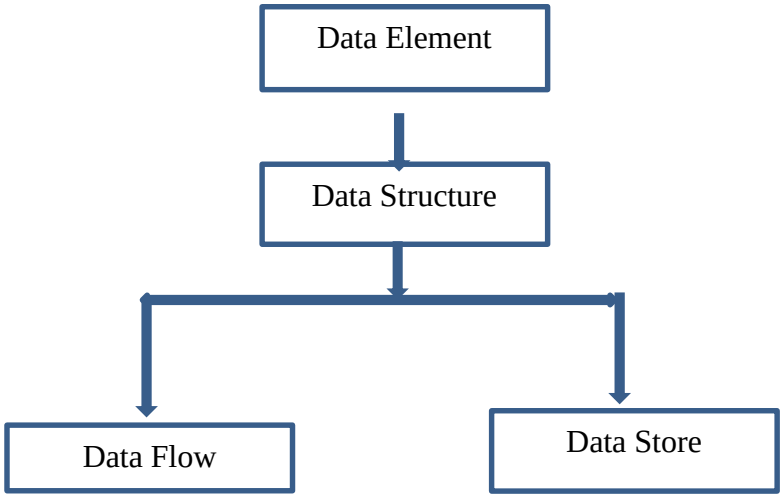


Fig 5.1 Logical data description hierarchy

5.1.1 Data Flow Diagram

5.1.1.1 Introduction

In the late 1970s dataflow diagram were introduced and popularized for structured analysis and design.

Definition

DFD is a means of representing a system at any level of detail with a graphic network of symbols showing data stores, data processes , and data sources/destinations. Data flow diagram model events and processes ie activities which transform data within system. DFDs examine how data flows into, out of , and within the system.

5.1.1.2 DFD Notations

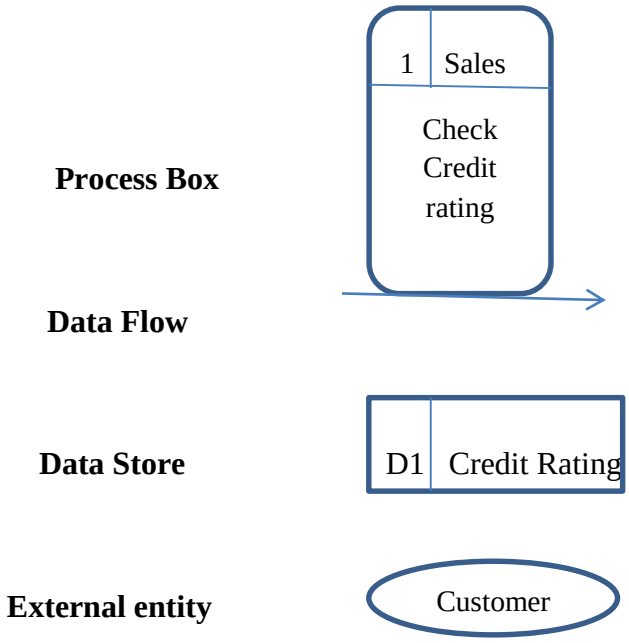
DFD are composed of the four basic symbols shown below:

a) **The Process** symbol represents an activity that transforms or manipulates the data. They may be shown as a circle, an oval, or a rectangle box

b) **The Data store** symbol represents data that is not moving.

c) **The Data flow** symbol represents movement of data. Data flow are generally shown as arrows coming to or going from the edge of a process box.

d) **The External Entity** symbol represents sources of data to the system or destination of data from the system.



5.2 DFD Notations

Any systems can be represented at any level of detail by these four symbols.

5.1.1.3 DFD Objectives

The purpose of DFD is to provide a semantic between users and system developers. The diagrams are

- Graphical: eliminating thousands of words
- Logical representations: modelling WHAT a system does, rather than physical models showing HOW it does it.
- Hierarchical : showing systems at any level of details.
- Jargonless: showing user understanding and reviewing

The DFD are basis of structured system analysis. Data flow diagrams are supported by other techniques of structured system analysis such as data structure diagrams, data dictionaries and procedure representing techniques such as decision tables, decision trees.

5.1.1.4 DFD Levels

In Software engineering DFD(data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher-level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD

- DFD Level 0 is also called a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

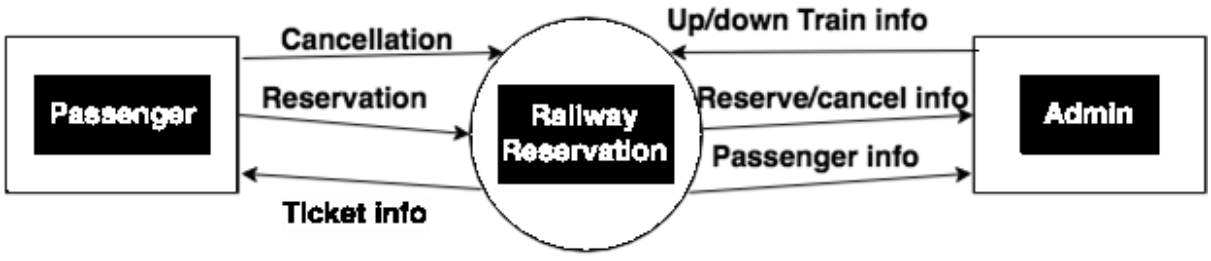


Figure 5.3 0 Level DFD

b) 1-level DFD

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

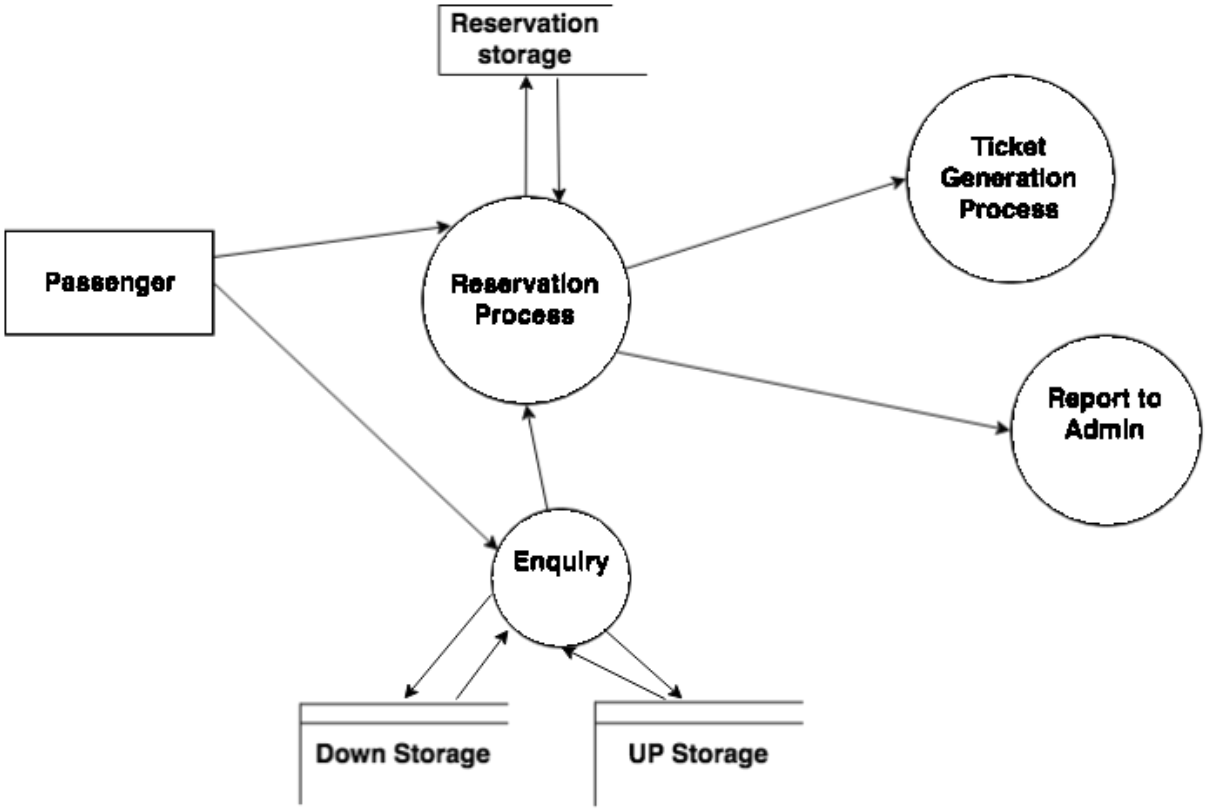


Figure 5.4 1-Level DFD

c) 2-level DFD

DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning. . It can be used to plan or record the specific/necessary detail about the system's functioning.

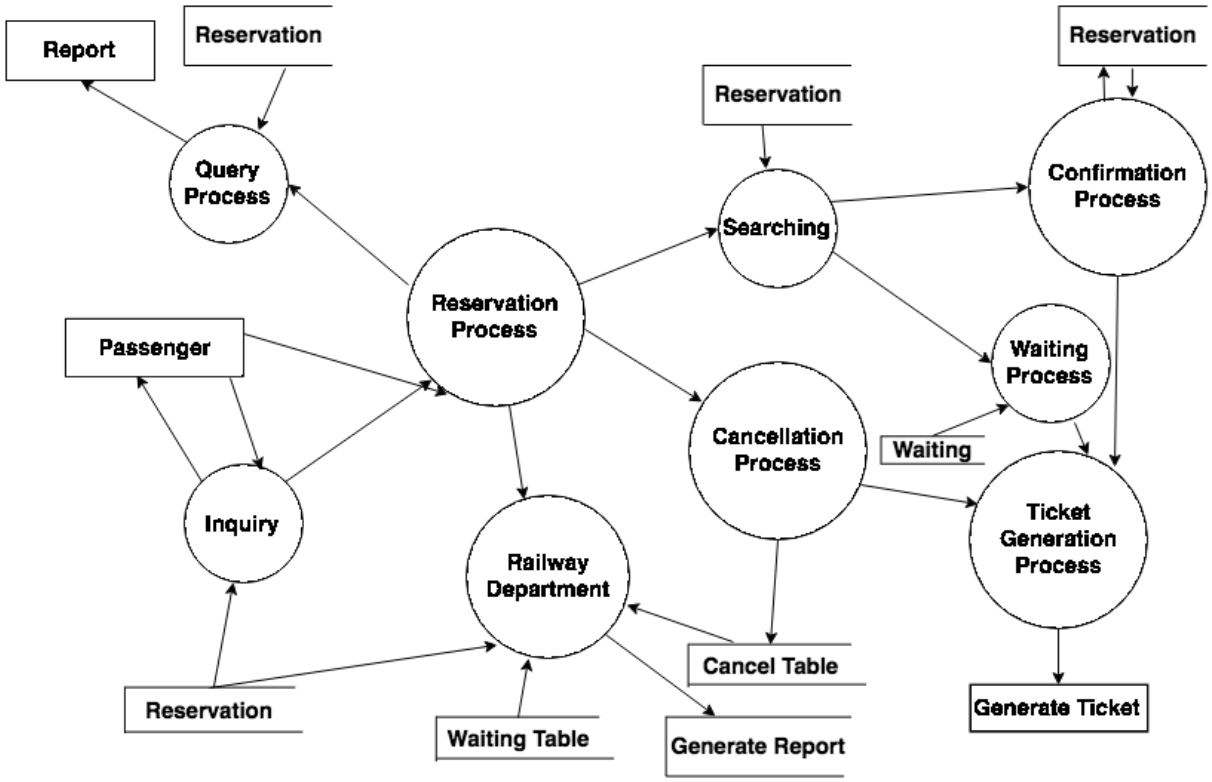


Figure 5.5 2-Level DFD

5.1.2 Decision Tables

A **Decision Table** is a tabular representation of inputs versus rules/cases/test conditions. It is a very effective tool used for both complex software testing and requirements management. Decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily. The conditions are indicated as True(T) and False(F) values.

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

5.1.2.1 Components of a Decision Table

a) **Condition Stub** − It is in the upper left quadrant which lists all the condition to be checked.

b) **Action Stub** − It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.

c) **Condition Entry** − It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.

d) **Action Entry** − It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

Decision table is typically divided into four quadrants, as shown below:

The Four Quadrants				
	Rule1	Rule2	Rule3	…..
Conditions Stub	Condition entries			
Actions Stub	Action entries			

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not satisfied.
- A blank - against action states it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table −

CONDITIONS	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N

Regular Customer	-	Y	N	-
ACTIONS				
Give 5% discount	X	X	-	-
Give no discount	-			

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to.

To make them more concise, many decision tables include in their condition alternatives a don't care symbol. This can be a hyphen or blank, although using a blank is discouraged as it may merely indicate that the decision table has not been finished. One of the uses of decision tables is to reveal conditions under which certain input factors are irrelevant on the actions to be taken, allowing these input tests to be skipped and thereby streamlining decision-making procedures.

Advantage of Decision Table:

Any complex business flow can be easily converted into the test scenarios & test cases using this technique.

Decision tables work iteratively that means the table created at the first iteration is used as input table for next tables. The iteration is done only if the initial table is not satisfactory.

Simple to understand and everyone can use this method design the test scenarios & test cases.

It provide complete coverage of test cases which help to reduce the rework on writing test scenarios & test cases.

These tables guarantee that we consider every possible combination of condition values. This is known as its completeness property.

Example 5.1

Consider a problem is to find largest of given three numbers x,y, and z. Construct decision table .

Solution

Decision table to determine largest

conditions								
x>y	**y**	**y**	**y**	**y**	**n**	**n**	**n**	**n**
x>z	**y**	**y**	**n**	**n**	**y**	**y**	**n**	**n**
y>z	**y**	**n**	**y**	**n**	**y**	**n**	**y**	**n**
actions								
Largest x	√	√	**-**				**-**	
Largest y			**-**		√	√	**-**	
Largest z			**-**	√			**-**	√

5.1.3 Decision tree

Decision table is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

A **decision tree** is a **decision** support tool that uses a **tree**-like model of **decisions** and their possible consequences.

A decision tree is a flowchart like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rule.

Decision tree is Graphical representation of the conditions, actions, and rules found

A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.

A decision tree consists of three types of nodes:

1. **Decision nodes** – typically represented by squares
2. **Chance nodes** – typically represented by circles
3. **End nodes** – typically represented by triangles

Decision trees can also be drawn with flowchart symbols, which some people find easier to read and understand.

Figure 5.6 illustrates general structure of decision tree

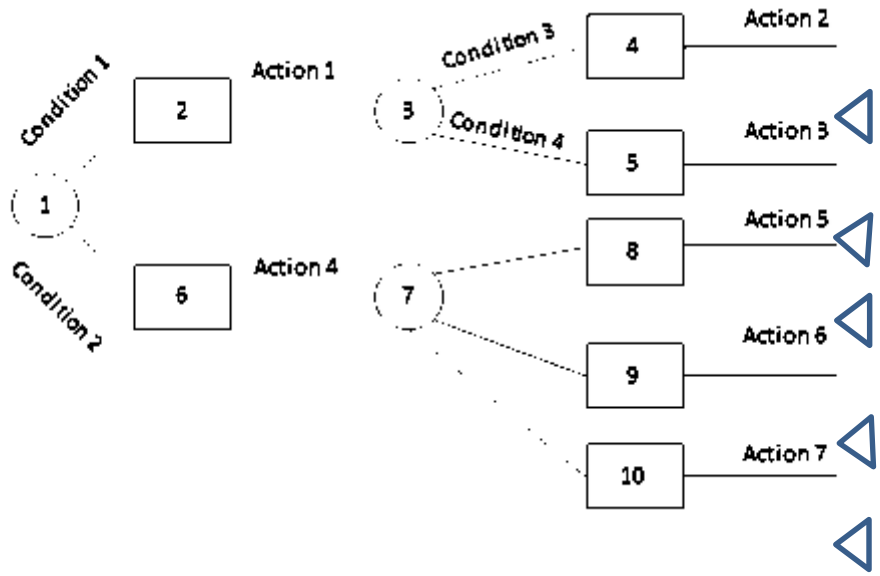


Figure 5.6 Structure of Decision Tree

5.1.3.1 Decision tree symbols

Shape	Name	Meaning
	**Decision node**	**Indicates a decision to be made**
	**Chance node**	**Shows multiple uncertain outcomes**

Shape	Name	Meaning
	**Alternative branches**	**Each branch indicates a possible outcome or action**
	**Rejected alternative**	**Shows a choice that was not selected**
	**Endpoint node**	**Indicates a final outcome**

The major limitation of a decision tree is that they are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal **decision tree**.

5.1.3.2 Advantages and disadvantages of Decision Tree

Advantages

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

Disadvantages

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.

- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

5.1.4 Data Dictionary

In DFD we give name to data flow, process , data store. although the names are descriptive of the data, the do not give detail. So a structured place is build to keep the detail, this place is called data dictionary. It's a rigorous definition of all DFD. A data dictionary improves the communication between the analyst and the user.

A **Data Dictionary** a set of information describing the contents, format, and structure of a database and the relationship between its elements, used to control access to and manipulation of the database.

It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

It maintains information about the definition, structure, and use of each data element that an organization uses.

Some **examples** of what might be contained in an organization's **data dictionary** include: ... The **data** types, e.g., integer, real, character, and image of all fields in the organization's data bases.

It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table −

Sr.No.	Roll NO	Name	Mark 1
1	8007	Ram	50

2	8005	Syam	60
3	8004	Sunder	80
4	8002	Ayush	55

Advantages of Data Dictionary

- It's works as a valuable reference.
- It improves the communication between analysis and user.
- it can be use to compare the data description.
- Data dictionary can be use for cross referenced.
- It's used in building a database.

5.1.5 Structured charts

Structure Chart in software engineering represents hierarchical structure of modules. It breaks down the entire system into lowest functional modules, describe functions and sub-functions of each module of a system to a greater detail. They are used in structured programming to arrange program modules into a tree.

Structured Charts are an example of a **top-down** design where a problem (the program) is broken into its components. The tree shows the relationship between modules, showing data transfer between the models.

Each module is represented by a box, which contains the module's name. The tree structure visualizes the relationships between modules, showing data transfer between modules using arrows. Modules at top level called modules at low level. Components are read from top to bottom and left to right. When a module calls another, it views the called module as black box, passing required parameters and receiving results

5.1.5.1 Symbols used in construction of structured chart

1. **Module**

   It represents the process or task of the system. It is of three types.

   - **Control Module**

     A control module branches to more than one sub module.

   - **Sub Module**

     Sub Module is a module which is the part (Child) of another module.

   - **Library Module**

     Library Module are reusable and invokable from any module.

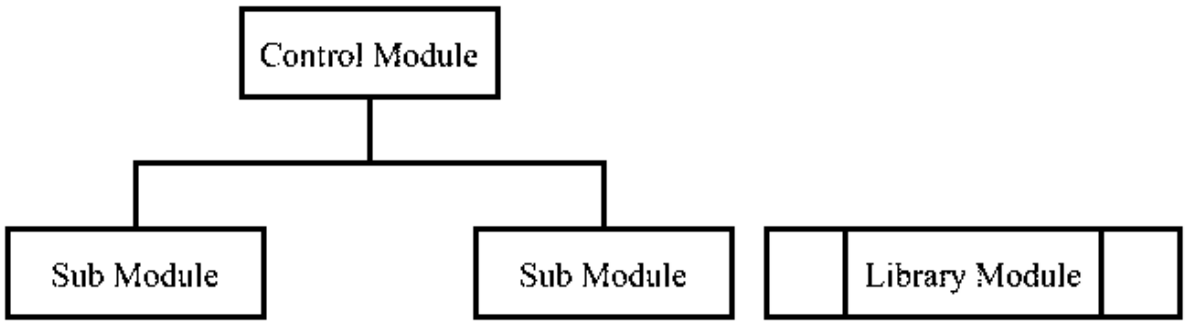


Figure 5.6 Process Modules

2. **Conditional Call**

   It represents that control module can select any of the sub module on the basis of some condition.

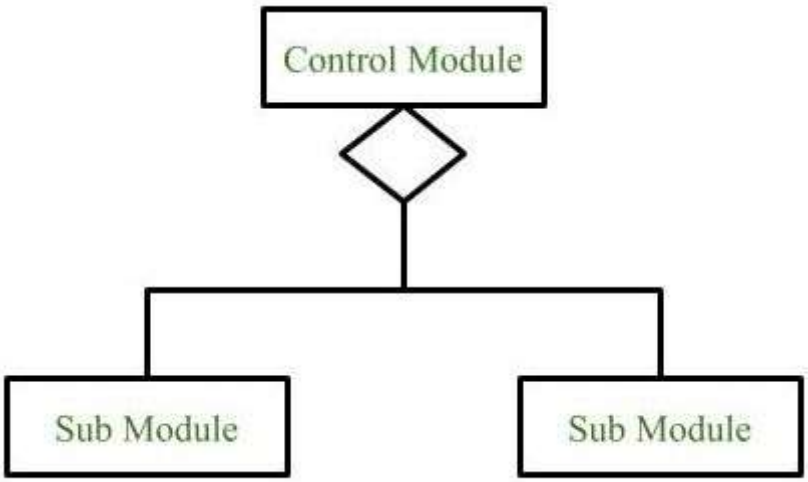


Figure 5.7 Conditional Module

3. Loop (Repetitive call of module)

It represents the repetitive execution of module by the sub module. A curved arrow represents loop in the module.

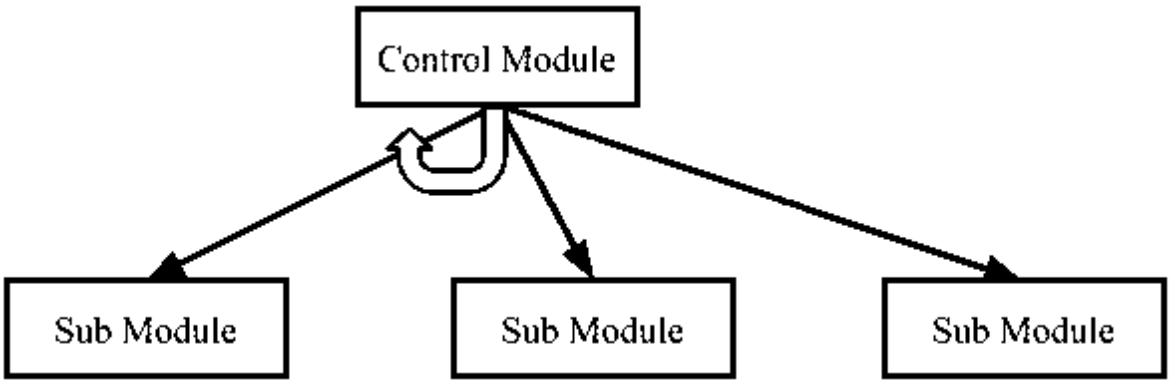


Figure 5.8 Loop Module

All the sub modules cover by the loop repeat execution of module.

4. Data Flow

It represents the flow of data between the modules. It is represented by directed arrow with empty circle at the end.

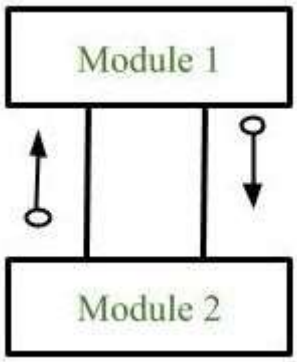


Figure 5.9 Data Flow

5. **Control Flow**

   It represents the flow of control between the modules. It is represented by directed arrow with filled circle at the end.

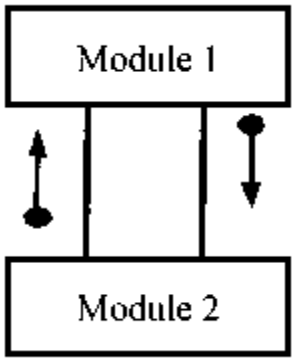


Figure 5.10 Control Flow

6. **Physical Storage**

   Physical Storage is that where all the information are to be stored.

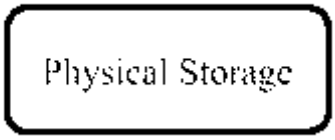


Figure 5.11 Physical Storage

Example 5.2

```
dim num1, num2 as integer
sub calculateAverage()
  dim avg as integer
  inputNums()
  avg = average(num1, num2)
  outputAvg(avg)
```

```
end sub

function average(a,b)
  return (a + b) / 2
end function

sub inputNums()
  num1 = console.readline()
  num2 = console.readline()
end sub


sub outputAvg(x)
  console.writeline("average = " & x)
end sub
```

Structure chart for code represented in example 5.2

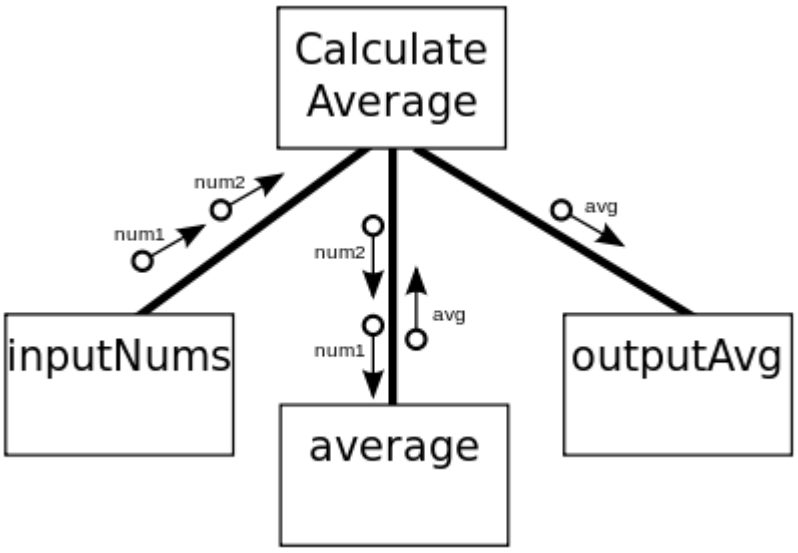


Figure 5.12 Structure chart for code represented tn Example 5.2

Example 5.3

Create structure charts for the following code:

```
sub howManyThrees()
  dim num1, count, total as integer
  num1 = startMsg()
  count = 0
  total = 0
  while num1 > 0 do
    checkNumber(count, total, num1)
    num1 = num1 - 1
  end while
  endMsg(count)
end sub
sub checkNumber(byRef c, byRef t, byVal n)
  If n MOD 3 = 0 Then
    c = divBy3(c)
  Else
    t = add(n, t)
  EndIf
end sub
function divBy3(x)
  return x + 1
end function
function add(n, t)
  return n + t
end function
function startMsg()
  console.writeline("program started, enter your number")
  return console.readline()
```

end function

sub endMsg(n)

console.writeline("number of threes : " & n)

end sub

Structure chart for code represented in example 5.3

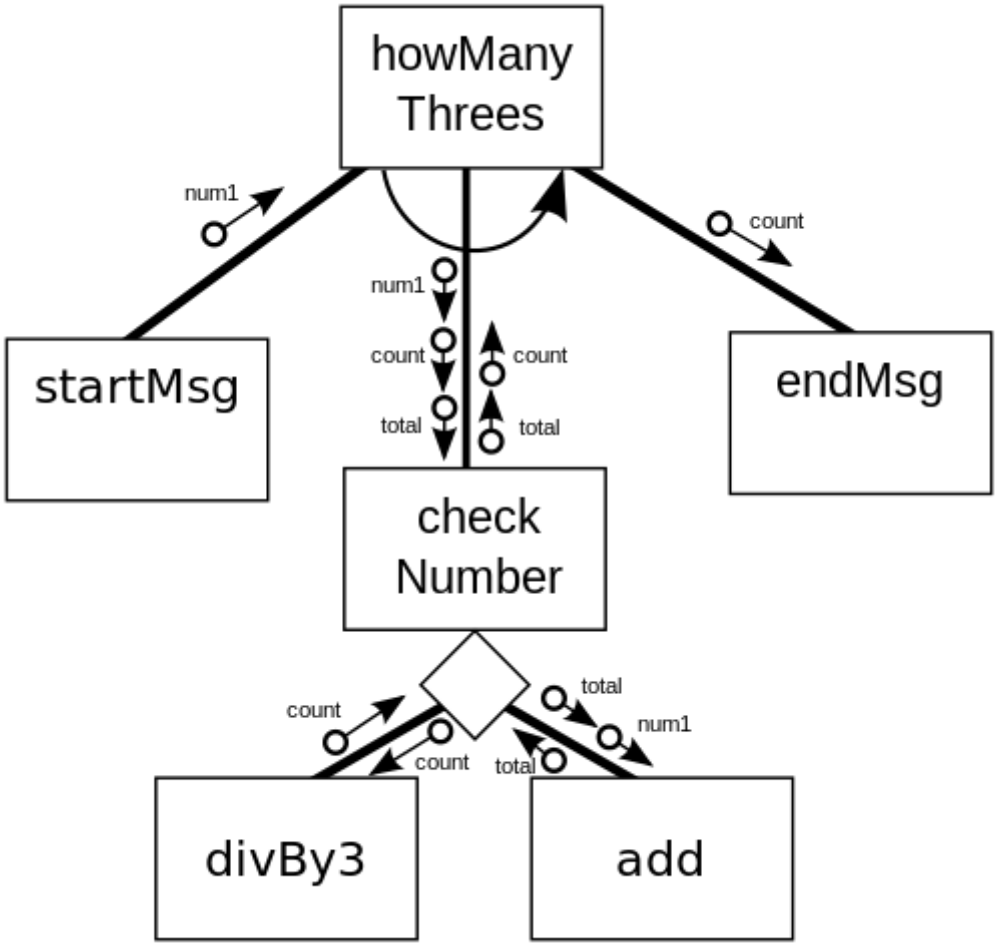


Figure 5.13 Structure chart for code represented tn Example 5.3

5.1.5.3 Comparison between Structure chart and Flowchart

Structure Chart	Flow Chart
Structure chart represents the software	Flow chart represents the flow of control in

architecture.	program.
It is easy to identify the different modules of the software from structure chart.	It is difficult to identify the different modules of the software from the flow chart.
Symbols used in structure chart are complex.	Symbols used in flow chart are simple.
Data interchange between different modules is represented here	Data interchange among different modules is not represented in flow chart.
In structure chart different types of arrows are used to represent data flow and module invocation.	Only a single type of arrow is used to show the control flow in flow chart.
Structure chart is hard to understand.	Flow chart is easy to understand.
Structure chart is complex to construct in comparison of flow chart.	Flow chart is easier to construct in comparison of structure chart.
It suppresses the sequential ordering of tasks inherent in a flow chart.	It demonstrates the sequential ordering of inherent tasks.

5.2 Object Oriented Analysis

Any software development approach goes through the following stages −

- Analysis
- Design
- Implementation

Phases in Object-Oriented Software Development

The major phases of software development using object–oriented methodology are

object-oriented analysis, object-oriented design, and object-oriented implementation.

5.2.1 Object–Oriented Analysis

In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real–world objects. The analysis produces models on how the desired system should function and how it must be developed. The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.

In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified.

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are

object modelling, dynamic modelling, and functional modelling.

5.2.1.1 Object Modelling

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modelling can be visualized in the following steps −

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes
- Review glossary

5.2.1.2 Dynamic Modelling

After the static behaviour of the system is analyzed, its behaviour with respect to time and external changes needs to be examined. This is the purpose of dynamic modelling.

Dynamic Modelling can be defined as "a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world".

The process of dynamic modelling can be visualized in the following steps −

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state–transition diagrams drawn

5.2.1.3 Functional Modelling

Functional Modelling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods. It specifies the meaning of the operations of object modelling and the actions of dynamic modelling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modelling can be visualized in the following steps −

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify constraints
- Specify optimization criteria

5.3 System Modeling

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML). • System modelling helps the analyst to

understand the functionality of the system and models are used to communicate with customers.

5.3.1 Context models

Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organisational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

5.3.2 Structural models

Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing. You create structural models of a system when you are discussing and designing the system architecture.

5.3.3 Behavioral models

Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.

You can think of these stimuli as being of two types:

– Data Some data arrives that has to be processed by the system.

– Events Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Self-Assessment Test

1. Compare structured analysis and Object-oriented analysis.
2. What do you meant by structured analysis? What are the various tools of Structured Analysis? Explain briefly.
3. Explain the concept of Object-oriented analysis. What are the various techniques of Object-oriented analysis? Explain briefly.
4. Compare structured chart and Flow chart.
5. Explain the importance of Decision table over Decision tree with a suitable example.

SUBJECT: Software Engineering	
COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 6	

Software Testing

Structure

6.0 OBJECTIVE

Objective of this lesson important of testing on software development . in this lesson various testing techniques and test case design techniques have been discussed when software is under development stage and software is ready for end user's. Further list of testing tools(static and dynamic) are discussed that are required for testing.

6..1 Introduction

Software Testing is the process of identifying the accuracy and quality of the software product and service under test. Apparently, it was born to validate whether the product fulfills the particular prerequisites, needs, and desires of the client. At the end of the day, testing executes a framework or application with a specific end goal to point out bugs, errors or defects. The responsibility of testing is to point out the issues of bugs and give developers a clue to help them fix it right following the requirements.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it's performing well and as per the specifications.

6.2 Some terminologies

In Software Testing, there is the difference between **Errors**, **Defects**, and **Bugs** that we should distinguish clearly to avoid misunderstanding problem.

Error : Error is a deviation from the actual and the expected result. It represents the mistakes made by people.

Bug : Bug is an error found BEFORE the application goes into production. A programming error that causes a program to work poorly, produce incorrect results, or crash. An error in software or hardware that causes a program to malfunction.

Defect: Defect happens once the error is identified during testing; it is logged as a 'Defect' in the tracking system.

Failures: Failure is the incapacity of a system to conduct its required functions within clarified performance requirements, literally a disappointment or a let-down. And no one wants to do business with a failure.

In short, "**A mistake in coding is called Error, Error found by the tester is called Defect, Defect accepted by development team then it is called Bug, build does not meet the requirements then it is a Failure**."

Figure 6.1 shows cues of software defects and relationship between them.

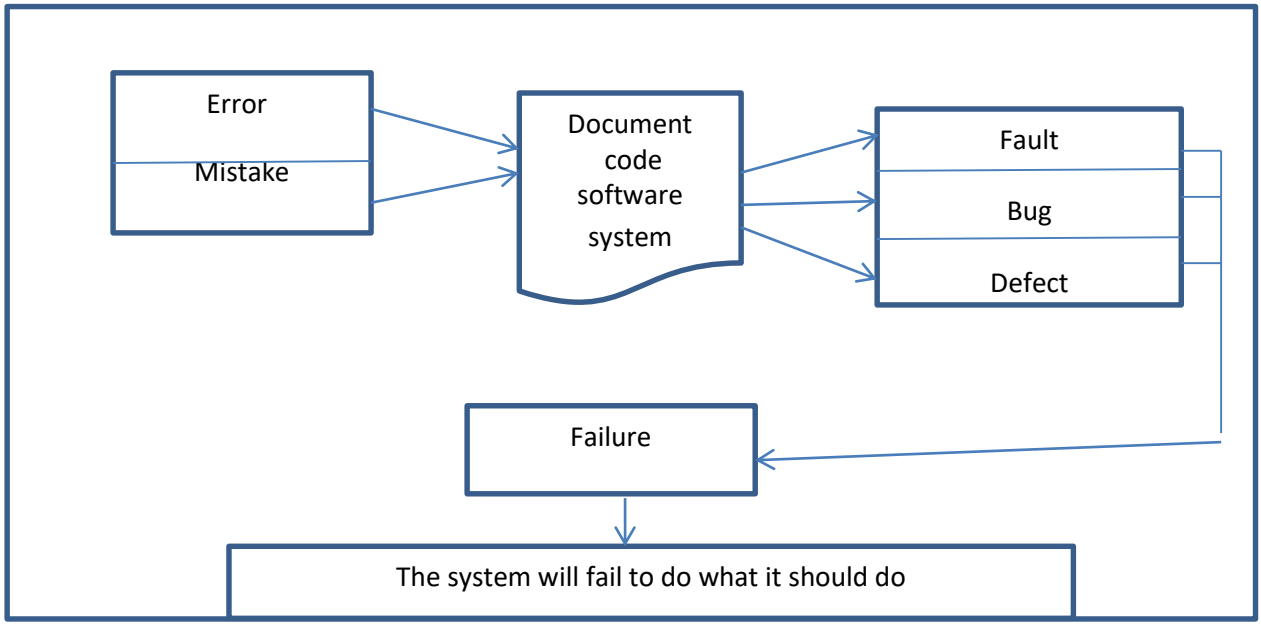


Figure 6.1 causes of Defects

6.3 Definitions of Testing

There are many definitions of testing. A few of them are given below:

I Testing is the process of demonstrating the errors are not present.

II The purpose of testing is to show that a program perform its tentened functions correctly.

III Testing is the process of establishing confidence that a program does what it is supposed to do.

IV Testing is the process of executing a program with the intent of finding faults. [MYER04]

V As per ANSI/IEEE 1059, Testing in Software Engineering is a process of evaluating a software product to find whether the current software product meets the required conditions or not.

VI Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free.

6.4 Types of Testing

Testing is an integral part of software development , it should not be considered aas distict phase of software development. A wide range of testing goes in a software development life cycle. Broadly , it can be categorised in to : .

Static testing: It is an important part testing. It has been claimed that as much as 70% to 80% of the software errors or faults can be identified early in the development process by static testing. Static testing includes Review, walkthrough, inspection performed during requirement analysis, design, and coded phases on non-executable products of the software development cycle.

Dynamic testing: it involves acual execution of the test cases according to the test procedures. , identification of failure, and resolution of module programme issues. There are two popular dynamic testing:

- Functional testing
- Non-functional testing

Figure 6.2 depicts categories of testing techniques

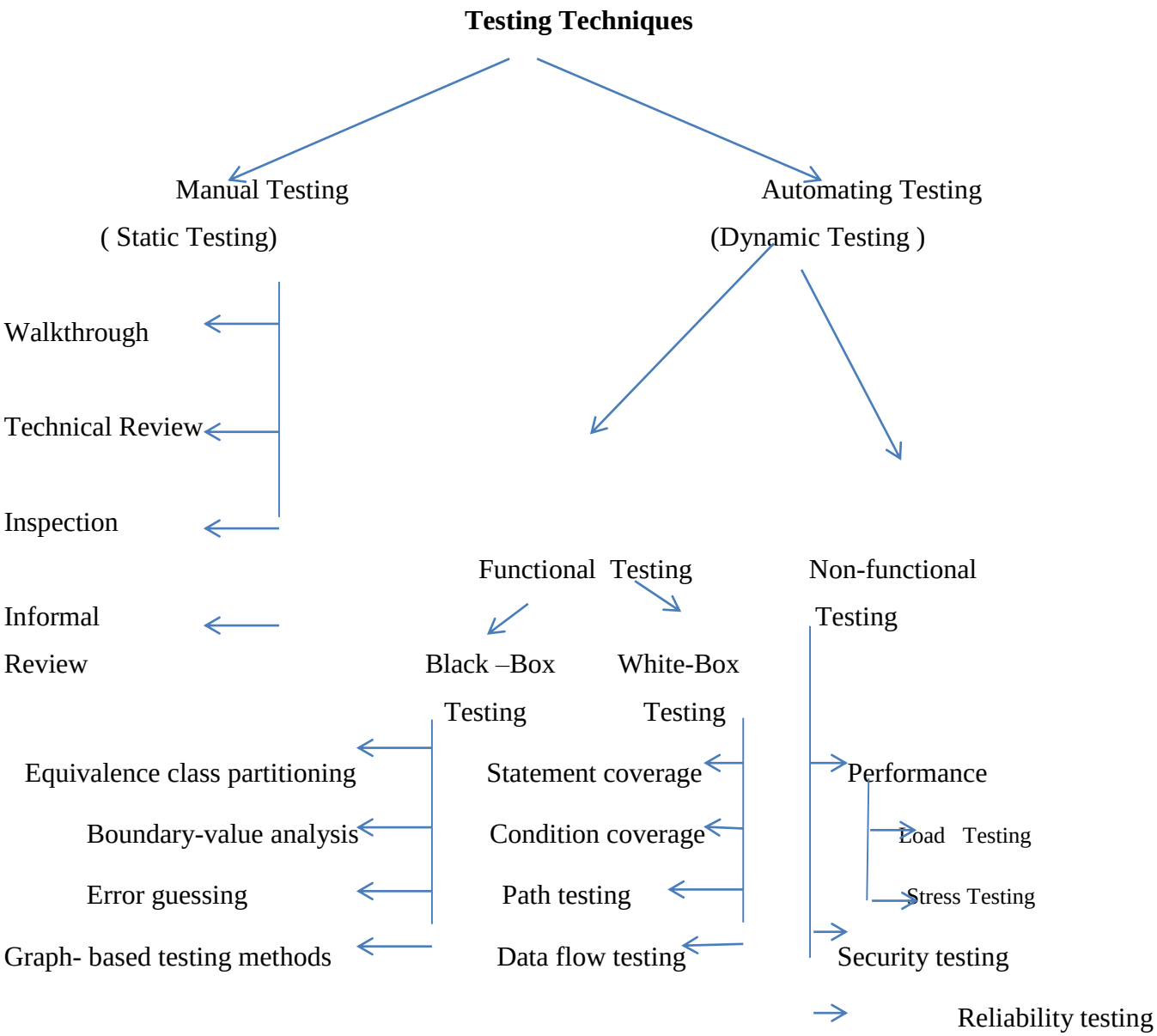


Figure 6.2 Testing Techniques

6.5 Testing Objectives

Software testing has different goals and objectives. The major objectives of Software testing are as follows:

- To gain the confidence of the customers by providing them a quality product.
- To check whether software which builds, it is as per the requirement or not.

- Finding defects from the software before customers find them out.
- Defects get a fix from the developer.
- Preventing defects

6.6 Testing and Inspection

6.6.1 Testing

Software testing is the process of evaluating the product that whether it's working properly as per specifications/requirements. It is related to finding bugs in UI, functionality and as per end-user perspective of the product

Any software built requires thorough testing to be successful in the market. If the built software does not meet the end user perspectives/SRS then it will not generate business and hence will not be effective. Testers working in software testing company that software is tested properly by considering all the requirements, use-cases, positive, negative and corner cases

TestinTypes:

Testing can broadly be divided into two categories:

1. Manual Testing: It is the core testing performed by testers to explore the product without using any automation tool.

2. Automation Testing: Here, test cases are automated using automation tool(like Selenium) and then run through script to save time.

Advantages of Software Testing:

Efficient software testing increases the following:

1. Market value of product
2. Product's credibility
3. Confidence
4. Efficiency

6.6.2 Inspection

Inspection is a disciplined practice for correcting defects in software artifacts**.**

Software inspection is testing plus code review to ensure that it is correct, optimized and maintainable. It is mainly related to finding bugs in program's code as per both requirements and test cases.

It is carried out to improve the processes and find the defects. Team schedules a meeting to discuss about the inspection process. In this meeting, team members are chosen to carry out the following 02 important roles:

1. **Moderator** : He takes in-charge of the meeting and leads the inspection process.

2. **Inspector** : He approves/disapproves the product after doing a thorough study and identifying bugs in it.

6.6.3.1 Steps in Inspection process:

1. Planning: Planning is the first step of everything. So moderator plans the inspection activities so that they are carried out smoothly.

2. Overview Meeting: It is carried out to spread knowledge regarding product's background.

3**. Inspection Meeting**: In this meeting, product documentation is read by Reader and Inspector makes efforts to find the defects.

4**. Rectification**: Required product changes are made based on bugs found out in inspection meeting.

5**. Follow-up Review**: Author reviews the changes made in product after rectification.

6.6.3.2 Types of Inspection:

1. Code Review: It is the process of inspecting a code block. Verification of code is done to check if it is according to the product functionality or performance improvements can be done.

2. **Peer Review**: It is a process where a number of people having domain specific knowledge and skills review the product. It helps in comprehensive checklist and reports of the product.

Advantages of Inspection:

1. New point of view: New defects can be found by those members who are new to software inspection.

2. **Sharing knowledge**: It is a great way to share knowledge regarding software design, functionality and finding bug methods in it.

3**. Finding defects early**: Finding defects early is a great boon if it is successful. Software inspection increases the probability of finding early defects with the help of various viewpoints.

6.7 Testing and Debugging

6.7.1 Testing − It involves identifying bug/error/defect in a software without correcting it. Normally professionals with a quality assurance background are involved in bugs identification. Testing is performed in the testing phase.

6.7.2 Debugging − It involves identifying, isolating, and fixing the problems/bugs. Developers who code the software conduct debugging upon encountering an error in the code. Debugging is a part of White Box Testing or Unit Testing. Debugging can be performed in the development phase while conducting Unit Testing or in phases while fixing the reported bugs

6.8 Software Testing Life Cycle (STLC)

6.8.1 Introduction

Software Testing Life Cycle (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities. Software Testing Life Cycle (STLC) is the testing process that is executed in a well-planned and systematic manner, which includes phases of the testing process. In the STLC process, various activities are carried out to improve the quality of the product. However, STLC phases only deal with testing and detecting errors but not development itself.

6.8.2 Software testing life cycle contains the following steps:

a) **Requirement Analysis**
b) **Test Planning**
c) **Test Case Development**
d) **Test Environment Setup**
e) **Test Execution**
f) **Test Cycle Closure**

Fig 6.3 shows Software Testing Life Cycle Model (STLC Model).

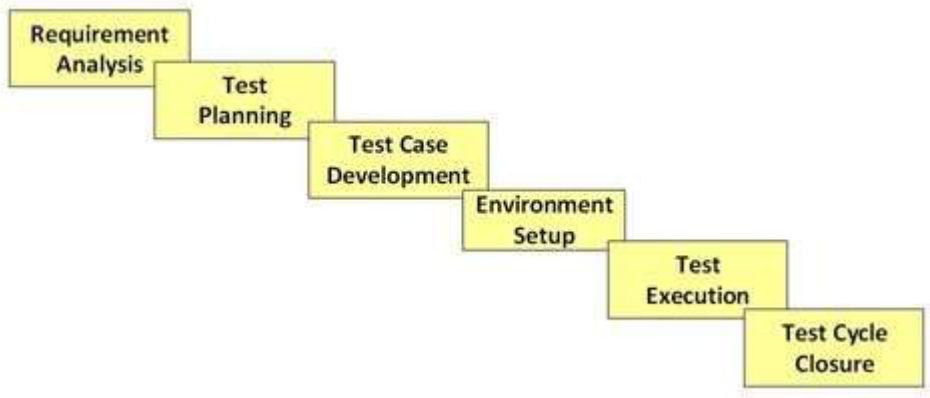


Figure 6.3 STLC Model

Each of these stages has a definite Entry and Exit criteria, Activities and Deliverables associated with it.

Entry Criteria: Entry Criteria gives the prerequisite items that must be completed before testing can begin.

Exit Criteria: Exit Criteria defines the items that must be completed before testing can be concluded

In an Ideal world, you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible. We will focus on activities and deliverables for the different stages in STLC life cycle.

a) Requirement Phase Testing

Requirement Phase Testing also known as Requirement Analysis in which test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail. Requirements could be either functional or non-functional.

In this phase, tester analyses requirement document of SDLC (Software Development Life Cycle) to examine requirements stated by the client. After examining the requirements, the tester makes a test plan to check whether the software is meeting the requirements or not.

b) Test Planning in STLC

Test plan creation is the crucial phase of STLC where all the testing strategies are defined. Tester determines the estimated effort and cost of the entire project. This phase takes place after the successful completion of the Requirement Analysis Phase. Senior QA manager responsible to determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined. The Test Plan gets prepared and finalized in the same phase.

c) Test Case Development Phase

The Test Case Development Phase involves the creation, verification and rework of test cases & test scripts after the test plan is ready. Initially, the Test data identified then created and reviewed and then reworked based on the preconditions. Then the QA team starts the development process of test cases for individual units. Test case execution can be started after the successful completion of Test Plan Creation.

d) Test Environment Setup

Test Environment Setup is an essential part of the manual testing procedure as without environment testing is not possible. Environment setup requires a group of essential software and hardware to create a test environment. The testing team is not involved in setting up the testing environment, its senior developers who create it. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase

e) Test Execution Phase

Test case Execution takes place after the successful completion of test planning. It is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed.

f) Test Cycle Closure

Test Cycle Closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members

meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

The test cycle closure report includes all the documentation related to software design, development, testing results, and defect reports.

This phase evaluates the strategy of development, testing procedure, possible defects in order to use these practices in the future if there is a software with the same specification

6.9 Software testing strategies

Unit testing

Integration testing

- **Top-down approach**
- **Bottom-up approach**
- **Sandwich(Hybrid) approach**

Acceptance/Validation testing

- **Alpha**
- **Beta**

System testing

- **Recovery testing**
- **Security testing**
- **Graphical user interface testing**
- **Compatible testing**

6.10 V-model/ V and V model /Verification and Validation model

In Waterfall fall model testing starts only after implementation is done. But if you are working in the large project, where the systems are complex, it's easy to miss out the key details in the requirements phase itself. In such cases, an entirely wrong product will be delivered to the client and you might have to start afresh with the project OR if you manage to note the requirements correctly but make serious mistakes in design and architecture of your software you will have to redesign the entire software to correct the error.

This model overcomes the drawback of the waterfall model. And in this model, testing starts from the requirement stage itself. V Model is a highly disciplined SDLC model in which there is a testing phase parallel to each development phase. The V model is an extension of the waterfall model in which testing is done on each stage parallel with development in a sequential way. It is known as the Validation and Verification Model.

In this model, first, all the activities go on the downward direction, and at one point in time, it starts moving in the upward direction to re-use the test document for the testing process and forms a V shape. Hence it is known as the V model.

We go for V and V model for the following reasons:

- For the large and complex application, here, large means that the n numbers of modules and complex specify lots of dependencies between modules.
- And It is also used for the long term projects.

Advantage and Disadvantage of V and V Model

Advantage	disadvantage
In this, review exists in every phase, that's why we may get less number of bugs in the application.	**It is a bit expensive process because Initial investment is high as the testing team is needed from the starting stage itself.**
The V model provides the Parallel deliverable, which implies that the two teams can work together like here; the development and testing team are working parallelly.	**It is a time-consuming process because if requirement changes happen, we need to change every text documents.**
This model helps to deliver Robust or stable products.	**In this, we need to do more documentation work because of the test cases and all other documents.**
In this model, the test Engineers	**The V model is not suitable for object-oriented projects.**

have more knowledge about the product because testing is involved in every stage of product development.	
The text document can be re-used.	**We cannot go back and replace the functionality once the application is in the testing phase.**

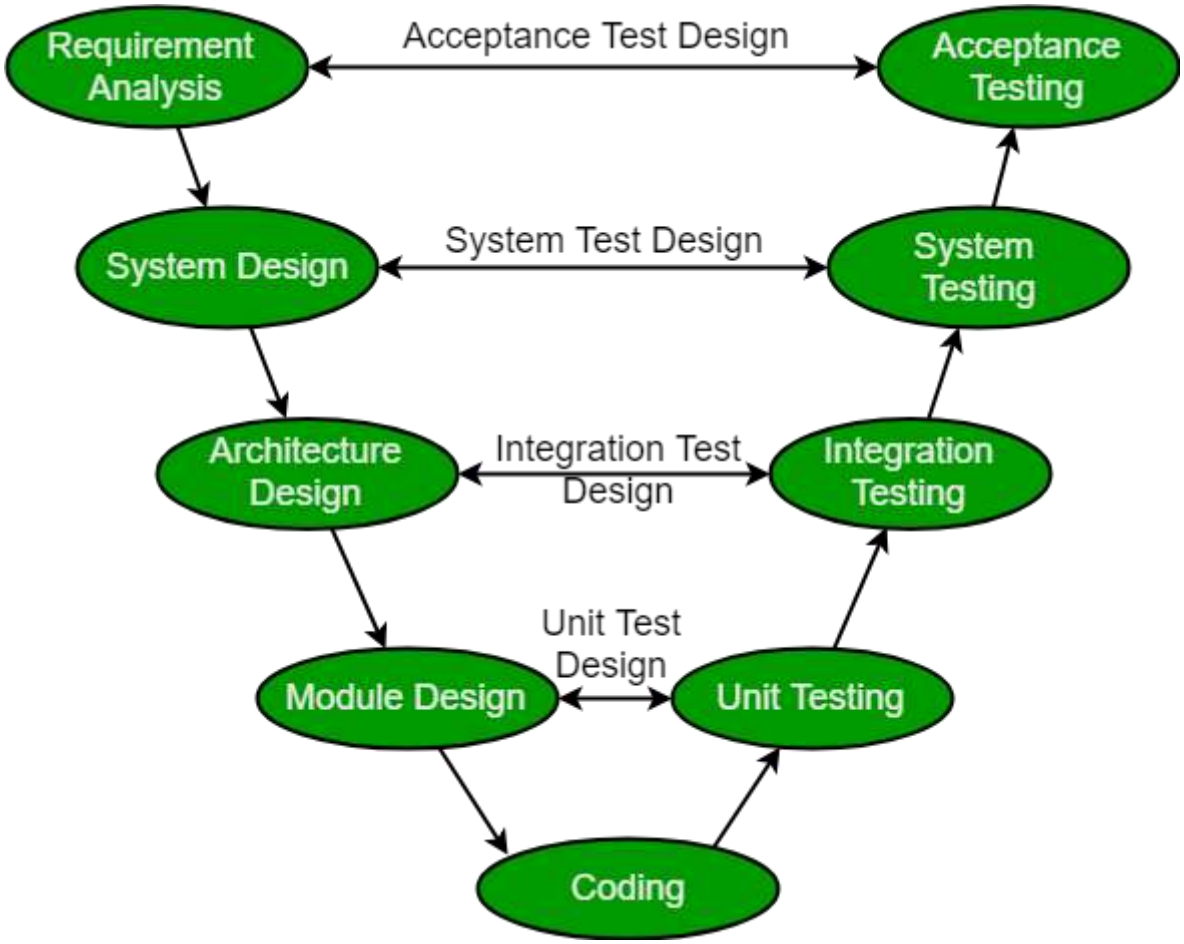


Figure 6.3 V-Model

6.10.1 V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

a) Business Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

b) System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

c) Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

d) Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

e) Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

6.10.2 Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

a) Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

b) Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

c) System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

d) Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

Difference between SDLC and STLC

Parameter	SDLC	STLC
Origin	Development Life Cycle	Testing Life Cycle
Objective	The main object of SDLC life cycle is to complete successful development of the software including testing and other phases.	The only objective of the STLC phase is testing.
Requirement Gathering	In SDLC the business analyst gathers the requirements and create Development Plan	In STLC, the QA team analyze requirement documents like functional and non-functional documents and create System Test Plan
High & Low-Level Design	In SDLC, the development team creates the high and low-level design plans	In STLC, the test analyst creates the Integration Test Plan
Coding	The real code is developed, and actual work takes place as per the design documents.	The testing team prepares the test environment and executes them
Maintenance	SDLC phase also includes post-deployment supports and updates.	Testers, execute regression suits, usually automation scripts to check maintenance code deployed.

6.11 Levels of testing

In general, there are four levels of testing: unit testing, integration testing, system testing, and acceptance testing. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

There are mainly four Levels of Testing in software testing :

i. **Unit Testing :** checks if software components are fulfilling functionalities or not.
ii. **Integration Testing** : checks the data flow from one module to other modules.
iii. **System Testing** : evaluates both functional and non-functional needs for the testing.

iv. **Acceptance Testing** : checks the requirements of a specification or contract are met as per its delivery.

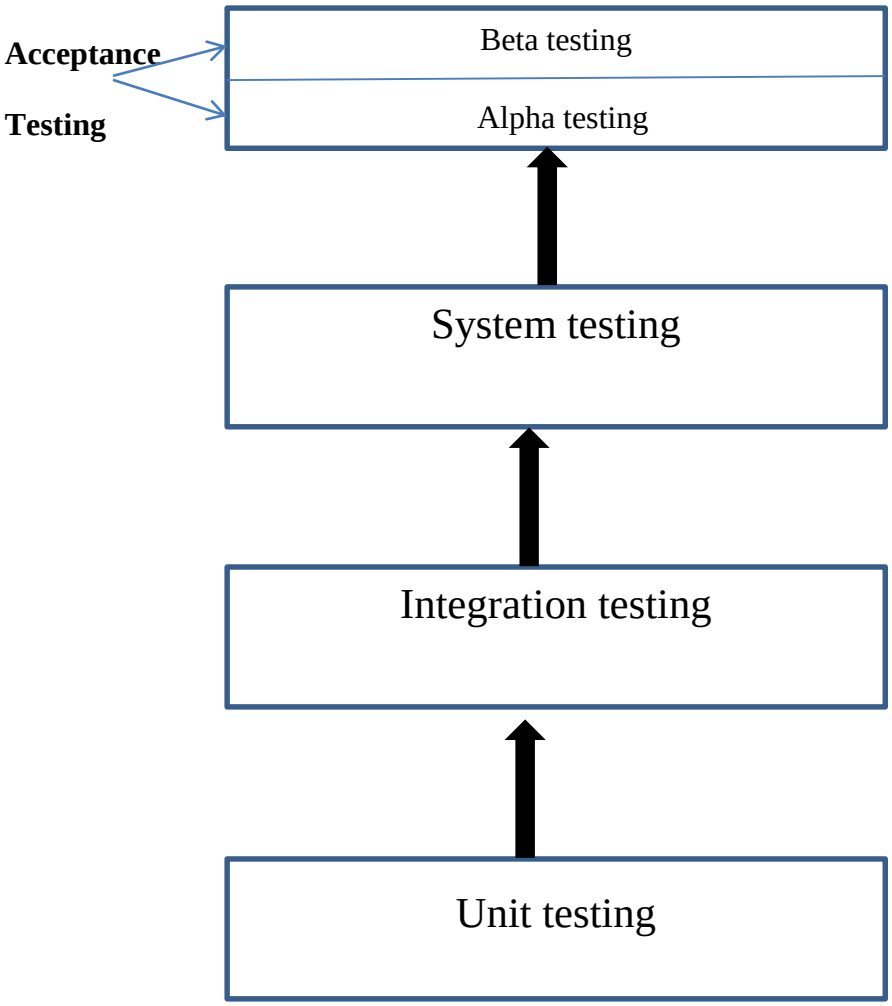


Figure 6.4 Levels of testing

Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

i) Unit Testing

Unit Testing is a type of software testing where individual units or components of software are tested. The purpose is to validate that each unit of the software code performs as expected.

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately. A unit may be an individual function, method, procedure, module, or object.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. Unit Testing is done during the development (coding phase) of an application by the developers. This kind of testing is performed by developers.

Unit Testing is of two types

- Manual
- Automated

Unit testing is commonly automated but may still be performed manually.

In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a WhiteBox testing technique that is usually performed by the developer.

Reason of Unit Testing?

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost defect fixing during system testm , integration testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

Here, are the key reasons to perform unit testing in software engineering:

Unit Testing Techniques

The Unit Testing Techniques are mainly categorized into three parts:

Black box testing that involves testing of user interface along with input and output

White box testing that involves testing the functional behaviour of the software application Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.

Code coverage techniques used in Unit Testing are listed below:

- Statement Coverage

- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

ii) Integration Testing

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'.

Approaches, Strategies, Methodologies of Integration Testing

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following
    - Top Down Approach
    - Bottom Up Approach
    - Sandwich Approach - Combination of Top Down and Bottom Up

Below are the different strategies, the way they are executed and their limitations as well advantages.

Big Bang Testing

Big Bang Testing is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.

Incremental Testing

In the Incremental Testing approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

- Bottom Up
- Top Down

Stubs and Drivers

Stubs and Drivers are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitutes for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.

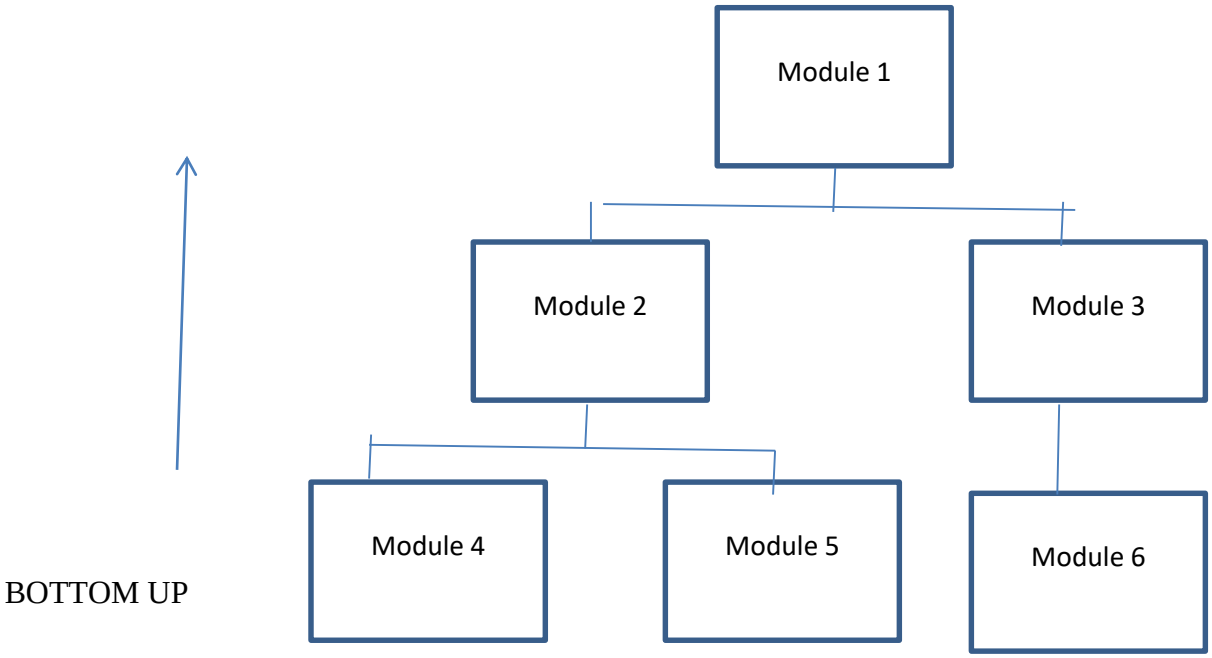
Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

Bottom-up Integration Testing

Bottom-up Integration Testing is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

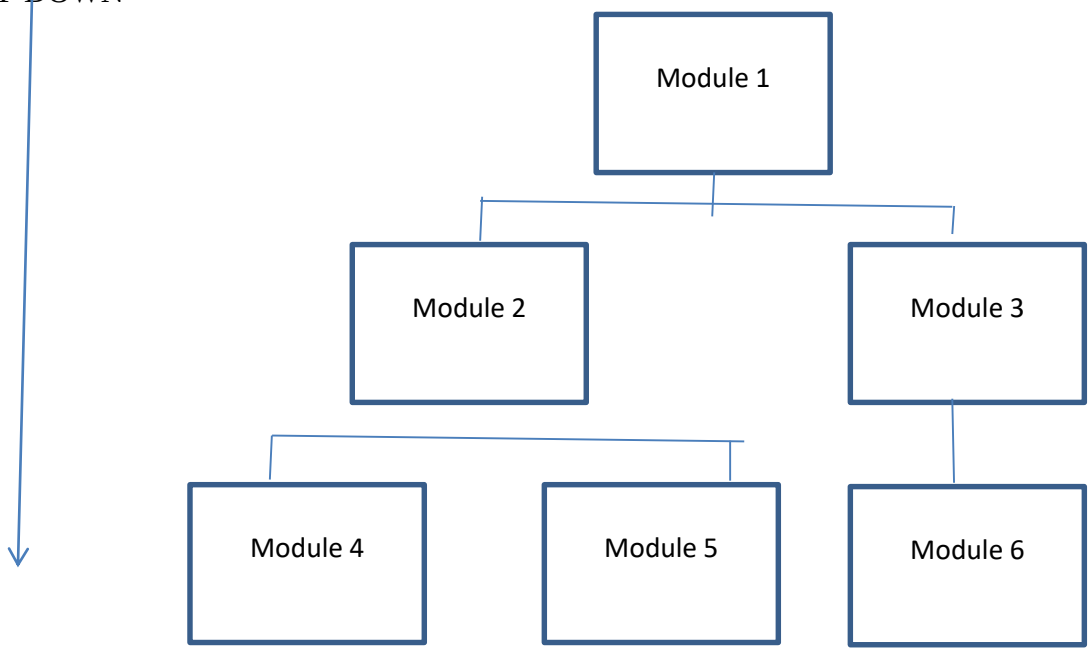
- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

Top-down Integration Testing

Top Down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

Diagrammatic Representation:

TOP DOWN



Advantages:

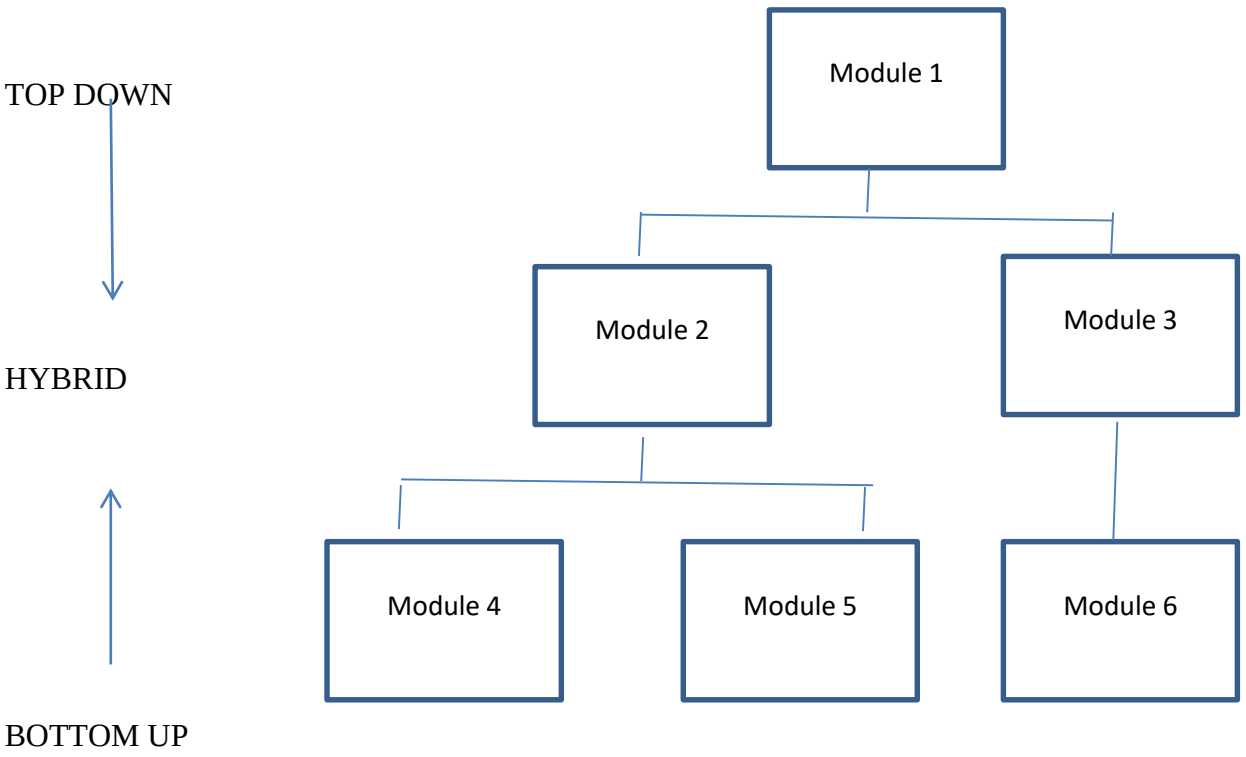
- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Sandwich Testing

Sandwich Testing is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called Hybrid Integration Testing. It makes use of both stubs as well as drivers.



Difference between unit testing and integration testing

Unit test	Integration test
• The idea behind Unit Testing is to test each part of the program and show that the individual parts are correct.	• The idea behind Integration Testing is to combine modules in the application and test as a group to see that they are working fine
• It is kind of White Box Testing	• It is kind of Black Box Testing
• It can be performed at any time	• It usually carried out after Unit Testing and before System Testing
• Unit Testing tests only the functionality of	• Integrating testing may detect errors when

the units themselves and may not catch integration errors, or other system-wide issues	modules are integrated to build the overall system
• It starts with the module specification	• It starts with the interface specification
• It pays attention to the behavior of single modules	• It pays attention to integration among modules
• Unit test does not verify whether your code works with external dependencies correctly.	• Integration tests verify that your code works with external dependencies correctly.
• It is usually executed by the developer	• It is usually executed by a test team
• Finding errors is easy	• Finding errors is difficult
• Maintenance of unit test is cheap	• Maintenance of integration test is expensive

iii) System Testing

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

Two Category of Software Testing

- Black Box Testing
- White Box Testing

System test falls under the black box testing category of software testing.

System testing done by a professional testing agent on the completed software product before it is introduced to the market.

iv) Acceptance Testing

Acceptance testing - beta testing of the product done by the actual end users.

Alpha Testing

Alpha Testing is a type of acceptance testing; performed to identify all possible issues and bugs before releasing the final product to the end users. Alpha testing is carried out by the testers who are internal employees of the organization. The main goal is to identify the tasks that a typical user might perform and test them.

To put it as simple as possible, this kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing. The main focus of alpha testing is to simulate real users by using a black box and white box techniques.

Beta Testing

Beta Testing is performed by "real users" of the software application in "real environment" and it can be considered as a form of external user acceptance testing . It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in customer's environment.

Alpha Testing	Beta Testing
Alpha testing performed by Testers who are usually internal employees of the organization	Beta testing is performed by Clients or End Users who are not employees of the organization
Alpha Testing performed at developer's site	Beta testing is performed at a client location or end user of the product
Reliability and security testing are not performed in-depth Alpha Testing	Reliability, Security, Robustness are checked during Beta Testing

Alpha testing involves both the white box and black box techniques	Beta Testing typically uses black-box testing
Alpha testing requires a lab environment or testing environment	Beta testing doesn't require any lab environment or testing environment. The software is made available to the public and is said to be real time environment
Long execution cycle may be required for Alpha testing	Only a few weeks of execution are required for Beta testing
Critical issues or fixes can be addressed by developers immediately in Alpha testing	Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product
Alpha testing is to ensure the quality of the product before moving to Beta testing	Beta testing also concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real time users.

Advantages of Alpha Testing:

- Provides better view about the reliability of the software at an early stage
- Helps simulate real time user behaviour and environment.
- Detect many showstopper or serious errors
- Ability to provide early detection of errors with respect to design and functionality

Advantages of Beta Testing

- Reduces product failure risk via customer validation.
- Beta Testing allows a company to test post-launch infrastructure.
- Improves product quality via customer feedback

- Cost effective compared to similar data gathering methods
- Creates goodwill with customers and increases customer satisfaction

Disadvantages of Alpha Testing:

- In depth, functionality cannot be tested as software is still under development stage Sometimes developers and testers are dissatisfied with the results of alpha testing

Disadvantages of Beta Testing

- Test Management is an issue. As compared to other testing types which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you seldom have control.
- Finding the right beta users and maintaining their participation could be a challenge

6.12 Regression Testing

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

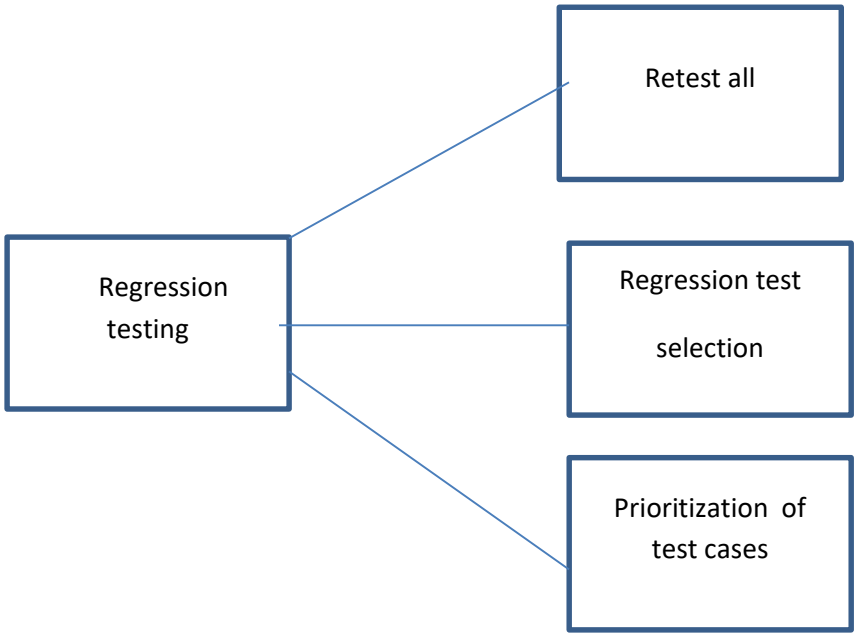
This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Need of Regression Testing

The Need of Regression Testing mainly arises whenever there is requirement to change the code and we need to test whether the modified code affects the other part of software application or not. Moreover, regression testing is needed, when a new feature is added to the software application and for defect fixing as well as performance issue fixing.

In order to do Regression Testing process, we need to first debug the code to identify the bugs. Once the bugs are identified, required changes are made to fix it, then the regression testing is done by selecting relevant test cases from the test suite that covers both modified and affected parts of the code.

Regression Testing can be carried out using the following techniques:



Retest All

- This is one of the methods for Regression Testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

Regression Test Selection is a technique in which some selected test cases from test suite are executed to test whether the modified code affects the software application or not. Test cases are categorized into two parts, reusable test cases which can be used in further regression cycles and obsolete test cases which can not be used in succeeding cycles.

Prioritization of Test Cases

- Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Selecting test cases for regression testing

It was found from industry data that a good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the test case for regression testing is an art and not that easy. Effective Regression Tests can be done by selecting the following test cases -

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

Difference between Re-Testing and Regression Testing:

Retesting means testing the functionality or bug again to ensure the code is fixed. If it is not fixed, defect needs to be re-opened. If fixed, Defect is closed.

Regression testing means testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.

An effective regression strategy, save organizations both time and money. As per one of the case study in banking domain, regression saves up to 60% time in bug fixes(which would have been caught by regression tests) and 40% in money

Retesting

Retesting is a process to check specific test cases that are found with bug/s in the final execution. Generally, testers find these bugs while testing the software application and assign it to the developers to fix it. Then the developers fix the bug/s and assign it back to the testers for verification. This continuous process is called Retesting.

What is Regression Testing?

Regression testing is a type of software testing executed to check whether a code change has not unfavourably disturbed current features & functions of an Application

Retesting vs Regression Testing

Regression Testing	Re-testing
• Regression Testing is carried out to confirm whether a recent program or code change has not adversely affected existing features	• Re-testing is carried out to confirm the test cases that failed in the final execution are passing after the defects are fixed
• The purpose of Regression Testing is that new code changes should not have any side effects to existing functionalities	• Re-testing is done on the basis of the defect fixes
• Defect verification is not the part of Regression Testing	• Defect verification is the part of re-testing
• Based on the project and availability of resources, Regression Testing can be carried out parallel with Re-testing	• Priority of re-testing is higher than regression testing, so it is carried out before regression testing
• You can do automation for regression testing, manual testing could be expensive and time-consuming	• You cannot automate the test cases for Retesting
• Regression testing is known as a generic testing	• Re-testing is a planned testing
• Regression testing is done for passed test cases	• Retesting is done only for failed test cases
• Regression testing checks for unexpected side-effects	• Re-testing makes sure that the original fault has been corrected

• Regression testing is only done when there is any modification or changes become mandatory in an existing project	• Re-testing executes a defect with the same data and the same environment with different inputs with a new build
• Test cases for regression testing can be obtained from the functional specification, user tutorials and manuals, and defect reports in regards to corrected problems	• Test cases for retesting cannot be obtained before start testing.

Regression Testing Tools

Following are the most important tools regression testing in software engineering:

- Selenium
- Quick Test Professional(QTP
- Rational Functional Tester (RFT

6.13 Smoke Testing

Smoke Testing is a software testing process that determines whether the deployed software build is stable or not. Smoke testing is a confirmation for QA team to proceed with further software testing. It consists of a minimal set of tests run on each build to test software functionalities. Smoke testing is also known as "Build Verification Testing" or “Confidence Testing.”

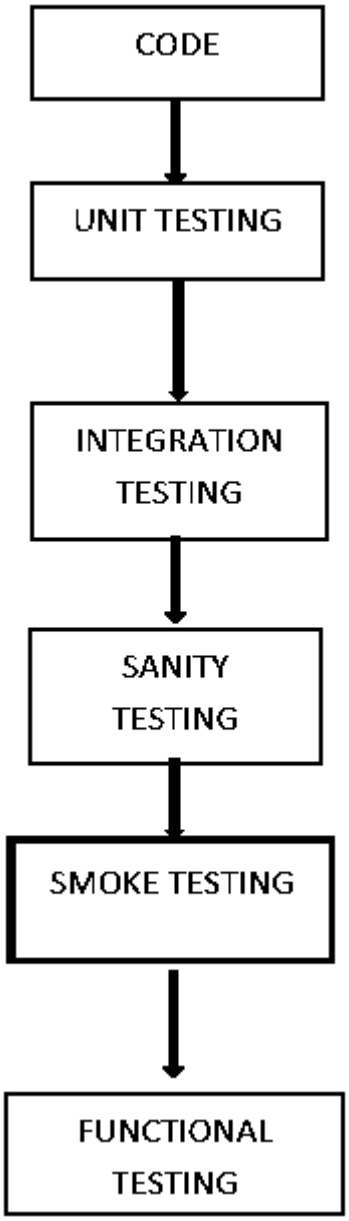
In simple terms, we are verifying whether the important features are working and there are no showstoppers in the build that is under testing.

It is a mini and rapid regression test of major functionality. It is a simple test that shows the product is ready for testing. This helps determine if the build is flawed as to make any further testing a waste of time and resources

Smoke Testing is a software testing technique performed post software build to verify that the critical functionalities of software are working fine. It is executed before any detailed functional or regression

tests are executed. The main purpose of smoke testing is to reject a software application with defects so that QA team does not waste time testing broken software application.

In smoke testing , the test cases chose to cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine. For Example, a typical smoke test would be - Verify that the application launches successfully, Check that the GUI is responsive ... etc.



Smoke Testing is done whenever the new functionalities of software are developed and integrated with existing build that is deployed in QA/staging environment. It ensures that all critical functionalities are working correctly or not.

In this testing method, the development team deploys the build in QA. The subsets of test cases are taken, and then testers run test cases on the build. The QA team test the application against the critical functionalities. These series of test cases are designed to expose errors that are in build. If these tests are passed, QA team continues with functional testing.

Any failure indicates a need to handle the system back to the development team. Whenever there is a change in the build, we perform Smoke Testing to ensure the stability.

Testing done in a development environment on the code to ensure the correctness of the application before releasing build to QA, this is known as Sanity testing. It is usually narrow and deep testing. It is a process which verifies that the application under development meets its basic functional requirements.

Sanity testing determines the completion of the development phase and makes a decision whether to pass or not to pass software product for further testing phase.

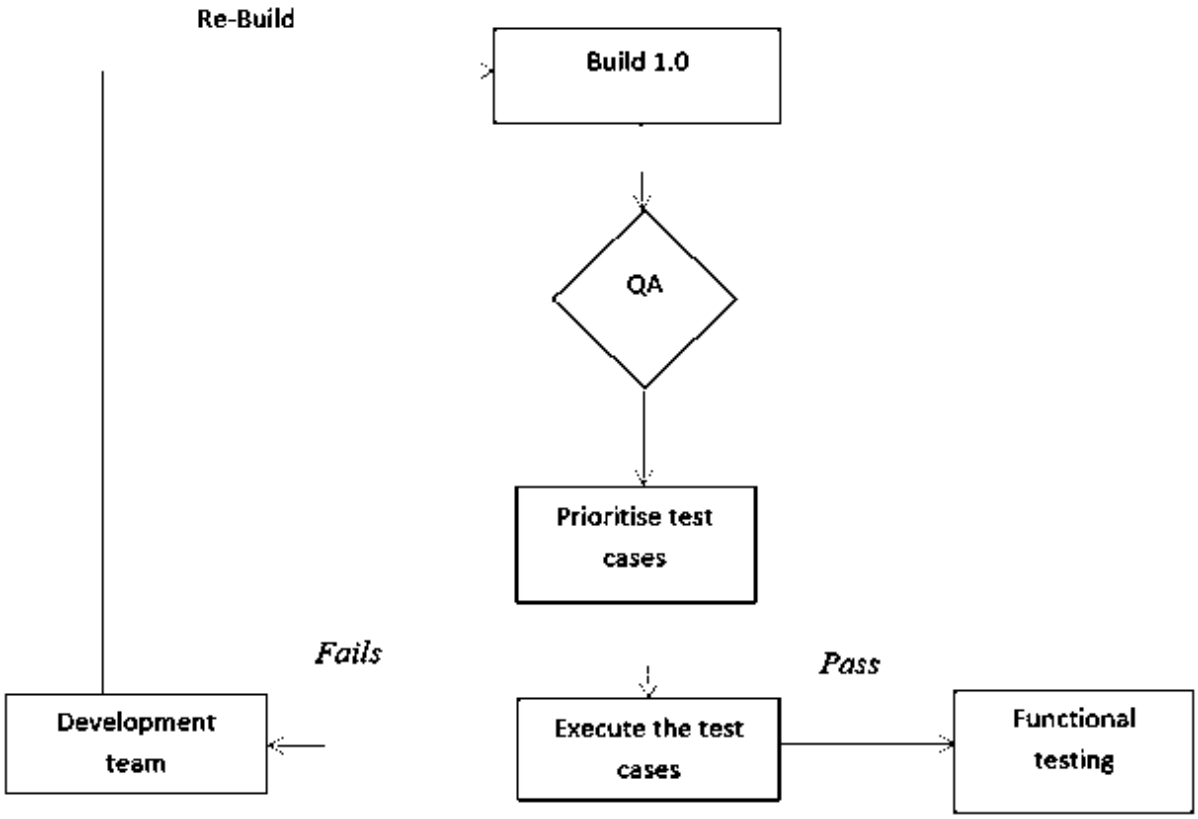
Smoke testing plays an important role in software development as it ensures the correctness of the system in initial stages. By this, we can save test effort. As a result, smoke tests bring the system to a good state. Once we complete smoke testing then only we start functional testing.

- All the show stoppers in the build will get identified by performing smoke testing.
- Smoke testing is done after the build is released to QA. With the help of smoke testing, most of the defects are identified at initial stages of software development.
- With smoke testing, we simplify the detection and correction of major defects.
- By smoke testing, QA team can find defects to the application functionality that may have surfaced by the new code.
- Smoke testing finds the major severity defects.

Smoke Testing is usually done manually though there is a possibility of accomplishing the same through automation. It may vary from organization to organization.

Smoke testing cycle

Below flow chart shows how Smoke Testing is executed. Once the build is deployed in QA and, smoke tests are passed we proceed for functional testing. If the smoke test fails, we exit testing until the issue in the build is fixed.



Advantages of Smoke testing

Here are few advantages listed for Smoke Testing.

- Easy to perform testing
- Defects will be identified in early stages.
- Improves the quality of the system
- Reduces the risk
- Progress is easier to access.
- Saves test effort and time
- Easy to detect critical errors and correction of errors.

- It runs quickly
- Minimises integration risks

If we don't perform smoke testing in early stages, defects may be encountered in later stages where it can be cost effective. And the defect found in later stages can be show stoppers where it may affect the release of deliverables.

6.14 System testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic System testing is actually a series of different testswhose primary purpose is to fully exercise the computerbased system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

Some of Different types of system testing are as follows:-

a) Recovery testing
b) Security testing
c) graphical user interface testing
d) Compatibility testing

a) Recovery Testing

Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properlyperformed. If recovery is automatic, re-initialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair is evaluated to determine whether it is
within acceptable limits.

Security Testing

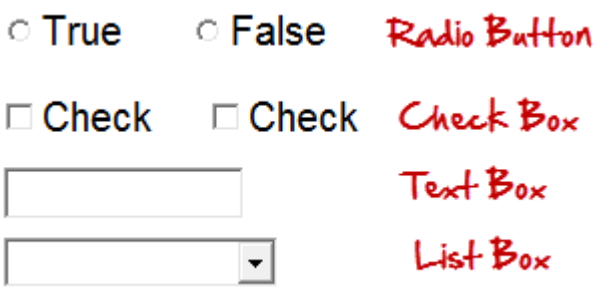
Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. During security testing, the tester plays the role(s) of the

individual who desires to penetrate the system. Anything goes! The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to breakdown any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to penetrate during recovery; may browse through insecure data, hoping to find the key to system entry.

GUI testing

There are two types of interfaces for a computer application. Command Line Interface is where you type text and computer responds to that command. GUI stands for Graphical User Interface where you interact with the computer using images rather than text.

Following are the GUI elements which can be used for interaction between the user and application:



GUI Testing is a validation of the above elements.

GUI Testing is a software testing type that checks the Graphical User Interface of the Software. The purpose of Graphical User Interface (GUI) Testing is to ensure the functionalities of software application work as per specifications by checking screens and controls like menus, buttons, icons, etc.

.Need of GUI Testing

Now the basic concept of GUI testing is clear. The few questions that will strike in your mind will be

- Why do GUI testing?
- Is it really needed?
- Does testing of functionally and logic of Application is not more than enough?? Then why to waste time on UI testing.

To get the answer to think as a user, not as a tester. A user doesn't have any knowledge about XYZ software/Application. It is the GUI of the Application which decides that a user is going to use the Application further or not.

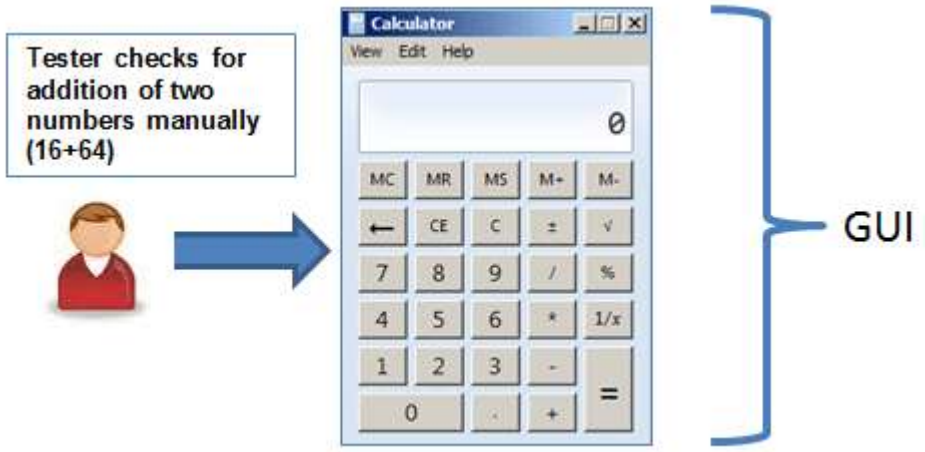
A normal User first observes the design and looks of the Application/Software and how easy it is for him to understand the UI. If a user is not comfortable with the Interface or find Application complex to understand he would never going to use that Application Again. That's why, GUI is a matter for concern, and proper testing should be carried out in order to make sure that GUI is free of Bugs.

I GUI Testing Techniques

GUI Testing Techniques can be categorized into three parts:

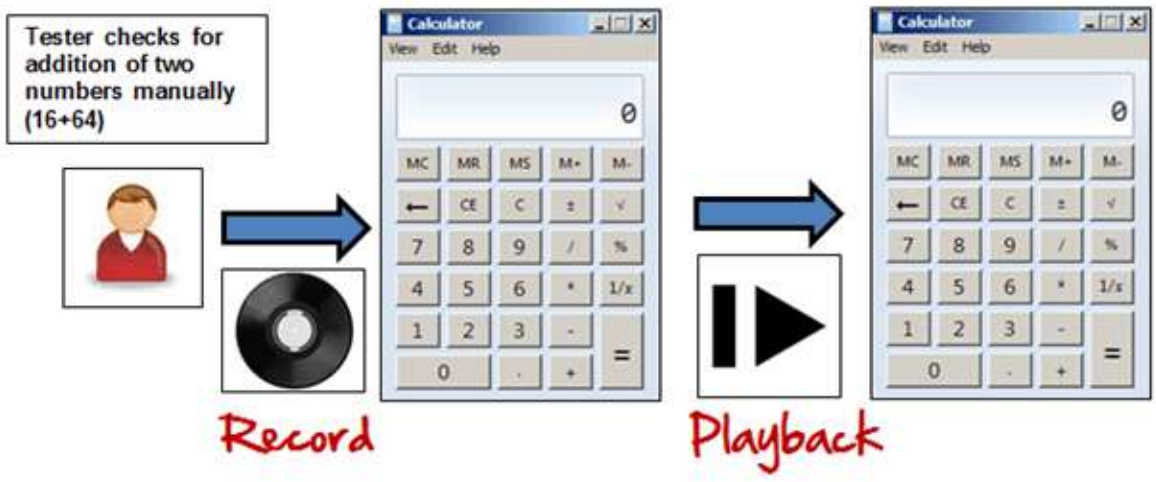
a) Manual Based Testing

Under this approach, graphical screens are checked manually by testers in conformance with the requirements stated in the business requirements document.

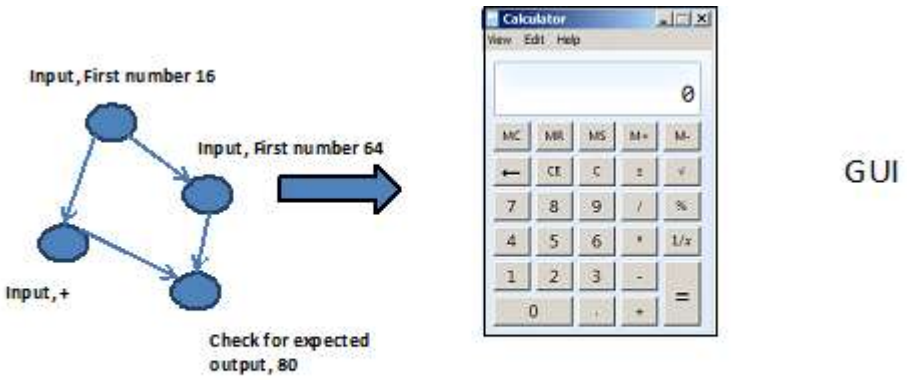


b) Record and Replay

GUI testing can be done using automation tools. This is done in 2 parts. During Record, test steps are captured by the automation tool. During playback, the recorded test steps are executed on the Application Under Test. Example of such tools - QTP.



Model Based Testing



Model Based Testing

Challenges in GUI Testing

In Software Engineering, the most common problem while doing Regression Testing is that the application GUI changes frequently. It is very difficult to test and identify whether it is an issue or enhancement. The problem manifests when you don't have any documents regarding GUI changes.

GUI Testing Tools

Following is a list of popular GUI Testing Tools :

- Selenium
- QTP
- Cucumber

- SilkTest
- TestComplete
- Squish GUI Tester

6.15 Performance testing

Performance Testing involve all the phases as the mainstream testing life cycle as an independent discipline which involve strategy such as plan, design, execution, analysis and reporting. Not all software has specification on performance explicitly. But every system will have implicit performance requirements. Performance has always been a great concern and driving force of computer evolution. The goals of performance testing can be performance bottleneck identification, performance comparison and evaluation. By performance testing we can measure the characteristics of performance of any applications. One of the most important objectives of performance testing is to maintain a low latency of a website, high throughput and low utilization.

Performance testing has two forms:-

6.15.1 Load testing

Load testing is the process of subjecting a computer, peripheral, server, network or application to a work level approaching the limits of its specifications. Load testing can be done under controlled lab conditions to compare the capabilities of different systems or to accurately measure the capabilities of a single system. In this we can check whether the software can handle the load of many user or not.

Load Testing is a non-functional software testing process in which the performance of software application is tested under a specific expected load. It determines how the software application behaves while being accessed by multiple users simultaneously. The goal of Load Testing is to improve performance bottlenecks and to ensure stability and smooth functioning of software application before deployment.

6.15.2 Stress testing

Stress testing is a testing, which is conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how.

Stress Testing is a type of software testing that verifies stability & reliability of software application. The goal of Stress testing is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn't crash under crunch situations. It even tests beyond normal operating points and evaluates how software works under extreme conditions.

In Software Engineering, Stress Testing is also known as Endurance Testing. Under Stress Testing, AUT is be stressed for a short period of time to know its withstanding capacity. A most prominent use of stress testing is to determine the limit, at which the system or software or hardware breaks. It also checks whether the system demonstrates effective error management under extreme conditions.

The application under testing will be stressed when 5GB data is copied from the website and pasted in notepad. Notepad is under stress and gives 'Not Responded' error message**.**

Need for Stress Testing

Consider the following scenarios -

- During festival time, an online shopping site may witness a spike in traffic, or when it announces a sale.
- When a blog is mentioned in a leading newspaper, it experiences a sudden surge in traffic.

It is imperative to perform Stress Testing to accommodate such abnormal traffic spikes. Failure to accommodate this sudden traffic may result in loss of revenue and repute.

Stress testing is also extremely valuable for the following reasons:

- To check whether the system works under abnormal conditions.
- Displaying appropriate error message when the system is under stress.
- System failure under extreme conditions could result in enormous revenue loss
- It is better to be prepared for extreme conditions by executing Stress Testing.

Goals of Stress Testing

The goal of stress testing is to analyze the behavior of the system after a failure. For stress testing to be successful, a system should display an appropriate error message while it is under extreme conditions.

Load Testing	Stress Testing
Load testing identifies the bottlenecks in the system under various workloads and checks how the system reacts when the load is gradually increased	Stress testing determines the breaking point of the system to reveal the maximum point after which it breaks.

To conduct Stress Testing, sometimes, massive data sets may be used which may get lost during Stress Testing. Testers should not lose this security-related data while doing stress testing.

The main purpose of stress testing is to make sure that the system recovers after failure which is called as recoverability.

Stress testing's objective is to check the system under extreme conditions. It monitors system resources such as Memory, processor, network etc., and checks the ability of the **system to** recover back to normal status. It checks whether the system displays appropriate error messages while under stress.

6.16 Object-oriented testing

Object-oriented programming concepts are different from traditional programming. We still do unit testing , integration testing , system testing to test the correctness of Object-oriented software. We also do regression testing in order to ensure changes have been implemented correctly.

Level of testing

In OO testing we have following levels of testing:
Method Testing(Unit Testing)
Class Testing(Unit Testing)
Inter-class Testing(Integration Testing)
System Testing

Integration Testing

It is also called ***inter-class testing*** . Due to absence of hierarchal control structure in OO programming following conventional integration testing techniques :

Top-down

Bottom –up

Sandwich

are not applicable .

In OO programming three popular techniques for inter-class testing

Thread based testing: in this testing technique classes are integrated that are needed to respond to an input given to system.

Use case based testing': in this where we combine clsses that are required by one use case.

are used

Cluster testing: where combine classes that are required to demonstrate one collaboration.

In all three approaches we combine classes on the basis of concept and execute them to see the outcome. Thread based testing is most popular due to simplicity .

OO Testing techniques

Path testing

State based testing

Class testing

Testing of class is very significant and critical

6.17 Software testing tools

The most important effort consuming task in software testing is to design the test cases. The execution of test cases may not require much time and resources.

Hence , the designing part is more significant than execution part. Both parts are normally handled manually.

Do we really need a tool! If yes where and when we can use it.

First part

Second part

Or both

Software testing tools may be used to reduce the time of testing and to make testing as easy. Automated testing may be carries out without human involvement. This may help users in the areas where a similar dataset is to given as input to perform again and again.

There broad two categories of testing tools

6.17.1 Static testing tools

Static software testing tools are those that perform analysis pof the program without execution them at all. These tools does not involves the actual input and output.

6.17.2 Dynamic testing tools

Dynamic testing tools select the test cases and execute the program to get the results . They also analyse the results and find reasons for failures(if any) of the program. These tools test the software with 'live' data.

Static testing is about preventation.

Dynamics testing is about cure.

We use both tools but preventation is always better than cure

Static testing tools find more bugs as compared to dynamic testing tools.

6.18 WEB Testing

Web applications are difficult ansd complex as compared to traditional client-servere application. A web application needs to be tested for:

a) Functional
b) Usability
c) Browser compatibility
d) Security
e) Load and stress
f) Storage and Database

a) Functional testing

Functional testing involves checking of specified functionality of a web application. The functional test cases for web applications may be generated using boundary value analysis, equivalence class testing, decision table, and many other techniques.

b) Usability testing

This testing technique tests that user interactions features. These features include hyperlinks, table forms, frames, and user interface items such as tecxt fields, radio buttons, check boxes, list boxes , command button, and dialog boxes.

c) Browser testing

Browser testing verifies the functioning of web application in terms of text, audio, vido and operating system corresponding to different browsers,. Browser compatibility matrix may be created to tst the web application on different browsers.

d) Security testing

Security testing requires an experienced tester having thorough knowledge of internet related security issues. The security expert needs to check security issues including authentications, unauthorised access, confidentiality, virus, firewalls and recovery from failure.

e) Load and stress

These are two parameters of performance testing.

The goal of performance testing is to evaluate the application performance with respect to real world scenario's. The following issues must keep in mind during performance testing

- Performance of system during peak hours(response time, reliability and availability)
- Points at which the system performance degrades or system fails.
- Impact of degraded performance on the customer loyalty, sales, and profits,

f) Data base testing

A data base must tested for admistrative level such as addition , deletion, updating an item in the database and user application searching. At this stage data base testing is plays important to ensure errors free web application.

SELF-ASSESSMENT TEST

2. Differentiate between Static and Dynamic testing .
3. Explain the following
    a) Regression testing
    b) Performance testing
    c) Stress testing
    d) GUI testing
4. Compare Black box testing and White box testing
5. What are the various levels of testing. Explain briefly. Why we require these levels?
6. What are the various testing strategies? Explain briefly.
7. What is Structural testing? What are the different types of structural testing techniques? Explain briefly.
8. Write a program to find roots of a quadratic equation.
   Draw flow graph . Also find independent paths.

9. Differentiate between Alpha testing and Beta testing .

10. .What are various levels of testing on oo testing? Which testing level is easy to test and why**?**

11. Explain V-model . What are the importance of V-Model in testing?

SUBJECT: SOFTWARE ENGINEERING	
COURSE CODE: MCA-14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 7	

SOFTWARE TESTING

STRUCTURE

7.0 LEARNING OBJECTIVE

This Chapter aims to provide a basic knowledge of testing. Some terminologies such as Test Artifacts, Test Plan, Test Case Design and Software Testing Strategies explain here. It also gives an overview of test process, V-model, various Levels of Software Testing and Software testing tools.

7.1 INTRODUCTION

Testing is a life-cycle activity which starts at the time of proposal and ends when acceptance testing is completed and product is finally delivered to customer. Testing begins with a proposal for software/ system application development/ maintenance, and ends when the system is formally accepted by user/customer. Different stakeholders are involved in conducting specialised testing required for specific application. It depends on the type of testing involved and level of testing to be done in various stages of software development life cycle. The main aim of testing is to maintain the quality of the product. Also, testing has its own cycle and in each phase, all focus revolves around quality only. Hence, testing refines the software and makes it as per the users expectations. Testing involves finding the difference between actual behaviours with respect to the expected behaviours of an application. Software testing involves verification as well as validation activities such as checking the compliance of the artifacts and activities with respect to defined processes and standards, and executing the software program to ensure that it performs correctly as desired by the customer and expressed in requirement agreed between development team and customer. There are many stages of software testing as per software development life cycle. It begins with feasibility testing at the start of the project, followed by contract testing and requirements testing, then goes through design testing and coding testing till final acceptance testing, which is performed by customer/user.

7.1.1 What is Testing?

Many people understand many definitions of testing**:**

1. Testing is the process of demonstrating that errors are not present.

2. The purpose of testing is to show that a program performs its intended functions correctly

3. Testing is the process of establishing confidence that a program does what it is supposed to do.

These definitions are incorrect. They describe almost the opposite of what testing should be viewed as.

"Software testing is the process of executing a program (or a part of program) with the intent of finding errors."

According to [DAHL72], software testing can't show the absence of errors, it can only show that errors are present. The essence of software testing is to determine a set of test cases for the software being tested. Effective software testing will contribute to the delivery of higher quality software products, more satisfied users, lower maintenance costs, more accurate, and reliable results.

7.1.2 Why is Software testing necessary?

Software testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – human make mistakes all the time.

Since we assume that our work may have mistakes, hence we all need to check our own work. However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done. Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

There are several reasons which clearly tell us as why Software Testing is important and what are the major things that we should consider while testing of any product or application.

Software testing is very important because of the following reasons:

1. To gain customer confidence
2. To check software adaptability
3. To identify errors
4. To avoid extra costs
5. To accelerate software development
6. To avoid risks

7. To optimize business

7.1.3 Objectives of Testing

To satisfy the definition of testing given earlier, testing must accomplish the following things.

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

7.1.4 Testing Principles

- All tests should be traceable to customer requirements.(from customers point of view)
- Tests should be planned long before testing begins. (Begins as soon as the requirements model is complete)
- The pareto principle applies to software testing. (implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components)
- Testing should begin “in the small” and progress toward testing “in the large.” (from individual components and ultimately in the entire system)
- Exhaustive testing in not possible. (It is not possible to exhaustively test every program path because the number of paths is simple too large)
- To be most effective, testing should be conducted by an independent third party. (Most effective means highest probability of finding errors)

7.1.5 Verification and validation

Verification and Validation are often used interchangeably but have different meaning. Verification refers to the set of activities that ensure that software correctly implements specific functions. It is the process of confirming that software meets its specification.

However, Validation is the process of confirming that software meets the customer's requirements. One is related to specification and other is related to customer requirements.

Verification: "Are we building the right product"

Validation: "Are we building the product right"

The verification and validation activities are combined under a broad term known as software testing.

Difference between verification and validation

- Verification process includes checking of documents, design, code and program whereas Validation process includes testing and validation of the actual product.
- Verification does not involve code execution while Validation involves code execution.
- Verification uses methods like reviews, walkthroughs, inspections and desk-checking whereas Validation uses methods like black box testing, white box testing and non-functional testing.
- Verification checks whether the software confirms a specification whereas Validation checks whether the software meets the requirements and expectations.
- Verification finds the bugs early in the development cycle whereas Validation finds the bugs that verification cannot catch.
- Comparing validation and verification in software testing, Verification process targets on software architecture, design, database, etc. while Validation process targets the actual software product.
- Verification is done by the QA team while Validation is done by the involvement of testing team with QA team.
- Comparing Verification vs. Validation testing, Verification process comes before validation whereas Validation process comes after verification.

7.2 SOME TERMINOLOGIES

7.2.1 Error, Mistake, Bug, Fault and Failure

People make errors. A good synonym is mistake. Error is basically difference between the actual output & the correct output. This may be a syntax error or misunderstanding of specifications. Sometimes, there are logical errors. When developers make mistake while coding, we call these mistakes "bugs".

An error may lead to one or more faults. It is more precise to say that a fault is the representation of an error. Fault is a condition that causes the software to fail. Defect is a good synonym for fault. If fault is in source code, we call it a bug. Failure is the inability of a system to perform a required function. It is produced only when there is a fault.

Note: presence of fault does not guarantee a failure.

7.2.2 Testing vs. Inspection

In simple words, Software testing is the process of evaluating the product that whether it's working properly as per requirements. It is related to finding bugs in UI, functionality and as per end-user perspective of the product while Software inspection is testing plus code review to ensure that it is correct, optimized and maintainable. It is mainly related to finding bugs in program's code as per both requirements and test cases.

7.2.3 Testing vs. Debugging

Software testing is a process of identifying defects in the software product. It is performed to validate the behavior of the software or the application compared to requirements.In other words, we can say that the testing is a collection of techniques to determine the accuracy of the application under the predefined specification but, it cannot identify all the defects of the software. Each software or application needs to be tested before delivering to the clients and checks whether the particular software or the application is working fine as per the given requirements.

As opposed to Testing, Debugging is the action where the development team or a developer implements after receiving the test report related to the bugs in the software from the testing team. In the software development process, it includes detecting and modifying code errors in a software program.

In debugging process, the developer needs to identify the reason behind the particular **bug** or defect, which is carried out by analyzing the coding rigorously. The developer changes the code and then rechecks whether the defect has been deleted whenever the bug or error is found.

Why do we need Debugging?

The process of debugging begins as soon as the **code** of the **software** is written. Then, it continues in successive stages as code is combined with other units of programming to form a software product. Debugging has many benefits such as:

- It reports an error condition immediately. This allows earlier detection of an error and makes the process of software development stress-free and unproblematic.
- It also provides maximum useful information of data structures and allows easy interpretation.
- Debugging assists the developer in reducing useless and distracting information.
- Through debugging the developer can avoid complex one-use testing code to save time and energy in software development.

Steps involved in Debugging

The different steps involved in the process of debugging are:

1. Identify the Error: A bad identification of an error can lead to wasted developing time. It is usual that production errors reported by users are hard to interpret and sometimes the information we receive is misleading. It is import to identify the actual error.

***2. Find the Error Location*:** After identifying the error correctly, you need to go through the code to find the exact spot where the error is located. In this stage, you need to focus on finding the error instead of understanding it.

3. Analyze the Error: In the third step, you need to use a bottom-up approach from the error location and analyze the code. This helps you in understanding the error. Analyzing a bug has two main goals, such as checking around the error for other errors to be found, and to make sure about the risks of entering any collateral damage in the fix.

4. Prove the Analysis: Once you are done analyzing the original bug, you need to find a few more errors that may appear on the application. This step is about writing automated tests for these areas with the help of a test framework.

5. Cover Lateral Damage: In this stage, you need to create or gather all the unit tests for the code where you are going to make changes. Now, if you run these unit tests, they all should pass.

6. Fix & Validate: The final stage is the fix all the errors and run all the test scripts to check if they all pass.

Both Testing and Debugging are the most important steps or practices during the development and after the development of any software or application developed in any programming language.

Some advantages of software testing are as below:

- It can easily understand by the new test engineers or the beginner.
- The test engineer can interact with software as a real end-user to check the usability and user interface issues.
- It is used to test dynamically altering GUI designs.
- Testing is a cost-effective and time-saving process.
- Software testing delivers consistence software.
- It will help us to execute the root cause analysis that will enhance the software's productivity.
- The testing process also helps detect and fixing the bugs before the software becomes active, which significantly reduces the risk of failure.

Some advantages of debugging process are as follows:

- It supports the developer in minimizing the data.
- If the perform the debugging, we can report the error condition directly.
- During the debugging process, the developer can avoid complex one-use testing code that helps the developer save time and energy.

- Debugging delivers maximum useful information of data structures and allows its informal understanding.

In the below table, we have listed some of the significant difference between testing and debugging:

S.NO	Testing	Debugging
1.	It is the implementation of the software with the intent of identifying the defects	The process of fixing and resolving the defects is known as debugging.
2.	Testing can be performed either manually or with the help of some automation tools.	The debugging process cannot be automated.
3.	A group of test engineers executes testing, and sometimes it can be performed by the developers.	Debugging is done by the developer or the programmer.
4.	The test engineers perform manual and automated test cases on the application, and if they detect any bug or error, they can report back to the development team for fixing.	The developers will find, evaluates, and removes the software errors.
5.	Programming knowledge is not required to perform the testing process.	Without having an understanding of the programming language, we cannot proceed with the debugging process.
6.	Once the coding phase is done, we proceed with the testing process.	After the implementation of the test case, we can start the Debugging process.
7.	Software Testing includes two or more activities such as validation and verification of the software.	Debugging tries to match indication with cause, hence leading to the error correction.

8.	It is built on different testing levels such as Unit Testing, Integration Testing, System Testing, etc.	It is built on different kinds of bugs because there is no such level of debugging is possible.
9.	Software testing is the presentation of defects.	It is a logical procedure.
10.	Software testing is the vital phase of SDLC (Software Development Life Cycle).	It is not a part of SDLC because it occurs as a subset of testing.

7.2.4 Test Artifacts

Test artifacts are also known as test deliverables. These are the reports or documents created while the testing is being carried out. These help in ensuring that the stakeholders are kept informed about the progress in the project. A software project undergoes several phases before it's finally delivered to the end-user. These phases are completely documented and presented to the client so that they are updated and can analyze if the requirements are being met or not.

There are various test artifacts created during an SDLC (Software development life cycle)

Some are delivered before the testing phase starts, some during the testing. While others after the whole project is developed.

Let us look at these artifacts one by one.

Types of Test Artifacts

1. ***Test Strategy***

   Test strategy is usually developed by the project manager. A test strategy is a document that lists the details about how the whole project will proceed. These include requirements and resources for the project, testing strategies involved, test design, incremental phases involved, client communication processes, etc. The document is just an outline that goes about how the whole project will be managed.

2. *Test Plan*

Test plan is often confused with a test strategy. It is a detailed document covering all the aspects of the testing phase. The test plan has a more systematic approach while covering minute details about how the whole testing phase will work. The document is dynamic and acts as a reference to map how the testing phase is progressing by the QA (Quality Assurance) team. A test plan includes details like-

- Scope of the project.
- The objective of the project.
- Resources required.
- Different effective testing approaches. How all the testing phases will be carried out?
- Risks involved.
- Actual and expected test results.
- Mapping out different phases of testing.
- The final table of failed and passed test results.

3. ***Test Scenario***

It is a condition created to perform successful end-to-end testing. Several test cases come under a test scenario. The test cases are developed on the basis of a high-level test scenario.

It is also sometimes referred to as a test condition or test possibility. A test scenario resolves the dilemma of software's real-life application.

4. ***Test Cases***

These are the extended part of the test scenario which helps in execution in testing. The document is quite detailed and includes information like steps to execute the test, test name, pre-and post-conditions of the tests, etc. They help to determine the functioning of the software in different situations.

Points that should be included in the test cases document are-

- Test case ID -Each test case has a unique identification number.

- Test case name.
- Description of the test case – It's an explanation of how the test case will proceed.
- Expected and actual outcomes of the test.
- Results – Either the test case is passed or failed.

5. *Test Data:*

   To run the test cases the QA engineer needs some data on which the test case is passed or failed is decide. For this, a document is prepared that contains all the test data required to run the created test cases.

6. *Required Traceability Matrix*

   The RTM is used to map out the many to many relationships between two documents. This is used to match the requirements of the client with the testing approach. It is maintained in a table format and links the requirements to verify if they are being fulfilled. There are two types of RTM- forward RTM and backward RTM. Details that are included in the traceability matrix document are-

- Requirement ID
- Requirement type or description.
- Test Case ID linked to that requirement.
- Result or the status. Whether the requirement was met or not.

7. Defect Report

A defect report is a document that is created once all the test cases are executed and the results are recorded. It enlists all the defects or bugs identified during the testing process. This makes the task to fix the bugs easier for the developer team.

8. Test summary report

At the end of each of the complete testing cycles, a final report is created. This report includes the details of the whole testing process. For example – details about test cases, the results obtained, bugs fixed, the current status of the project, etc.

This document is important and has to be presented to the stakeholders to keep them informed about the progress made in the project. This is a formal document and everything is enlisted clearly so that anyone can understand the report.

9. User guide

Once the whole software is developed and is ready to be deployed into the market, a user guide is created in the end. This guide is helpful to the end-users and gives detailed information about the software and its usage.

7.2.5 Test Case Design

Test case design refers to how you set-up your test cases. It is important that your tests are designed well, or you could fail to identify bugs and defects in your software during testing. Designing good test cases ensure that every aspect of your software gets tested so that you can find and fix any issues. It defines how documents are written and interpreted by each person involved in software development life cycle.

Following are the typical test case design techniques in software engineering:

1. Deriving test cases directly from a requirement specification or black box test design technique. The Techniques include:

- Boundary Value Analysis (BVA)
- Equivalence Partitioning (EP)
- Decision Table Testing
- State Transition Diagrams
- Use Case Testing

2. Deriving test cases directly from the structure of a component or system:

- Statement Coverage
- Branch Coverage
- Path Coverage

- Decision Coverage

3. Deriving test cases based on tester's experience on similar systems or testers intuition:

- Error Guessing
- Exploratory Testing

7.3 Software Testing Life Cycle

Software Testing Life Cycle (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities. Contrary to popular belief, Software Testing is not just a single/isolate activity, i.e. testing. It consists of a series of activities carried out methodologically to help certify your software product. STLC stands for Software Testing Life Cycle.

Software testing life cycle phases

There are following six major phases in every Software Testing Life Cycle Model (STLC Model):

1. Requirement Analysis
2. Test Planning
3. Test case development
4. Test Environment setup
5. Test Execution
6. Test Cycle closure

1. Requirement Phase Testing

Requirement Phase Testing also known as Requirement Analysis in which test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail. Requirements could be either functional or non-functional. Automation feasibility for the testing project is also done in this stage.

Activities in Requirement Phase Testing are:

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM)
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

Deliverables of Requirement Phase Testing

- RTM
- Automation feasibility report. (if applicable)

2. Test Planning in STLC

Test Planning in STLC is a phase in which a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined. The Test Plan gets prepared and finalized in the same phase.

Activities in Test Planning are:

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement

Deliverables of Test Planning

- Test plan /strategy document.
- Effort estimation document.

3. Test Case Development Phase

The **Test Case Development Phase** involves the creation, verification and rework of test cases & test scripts after the test plan is ready. Initially, the Test data is identified then created and reviewed and then reworked based on the preconditions. Then the QA team starts the development process of test cases for individual units.

Activities in Test Case Development are:

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

Deliverables of Test Case Development

- Test cases/scripts
- Test data

4. Test Environment Setup

Test Environment Setup decides the software and hardware conditions under which a work product is tested. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase. Test team may not be involved in this activity if the development team provides the test environment. The test team is required to do a readiness check (smoke testing) of the given environment.

Activities in Test Environment Setup are:

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

Deliverables of Test Environment Setup

- Environment ready with test data set up

- Smoke Test Results.

5. Test Execution Phase

Test Execution Phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed.

Activities in **Test Execution are:**

- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the defect fixes
- Track the defects to closure

Deliverables of Test Execution

- Completed RTM with the execution status
- Test cases updated with results
- Defect reports

6. Test Cycle Closure

Test Cycle Closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

Activities in Test Cycle Closure are:

- Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.

- Document the learning out of the project
- Prepare Test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

Deliverables of Test Cycle Closure

- Test Closure report
- Test metrics

7.4 Software Testing Strategies and Techniques

A test strategy is a concise statement that describes how the objectives of the software testing are met. The test strategy views the test event at a high level, concentrates on the objectives of the test event, the techniques that can be used and the resources that are required. It is vital that developers should plan their approach to testing at every stage of the project and establish a framework for testing the project. A testing technique can be defined as a process that ensures that the application being tested functions in a structured way. A testing strategy and technique is based on the method of testing adopted. Let us now learn the different methods of testing.

The three main methods of testing are the following:

1. Structural vs. functional testing

2. Static vs. dynamic testing

3. Manual vs. automated testing

7.4.1 Structural versus Functional Testing

If the test cases are developed to check the actual structure of the program code, then it is called structural testing. Structural testing is also known as white box testing, where the tester checks the actual code of the software. However, in functional testing, the tester checks only the behavior of the software and will not check the actual code. The tester only checks the response of the software for predefined inputs and tests whether the software produces the desired output. Therefore, this is called

black box testing. If the test cases are developed to check how the entire system works, then it is called functional testing. During functional testing, the tester uses test cases to check how the software works, i.e., whether it produces the desired outputs for a set of given inputs. Functional testing for a calculator could check whether the software does the addition operation correctly. We will cover Structural and Functional Testing in detail in unit 3.

7.4.2 Static versus Dynamic Testing

Static testing refers to the analysis of the program, which is carried out without executing the program. This is a typical white box testing technique, where the developer checks the code to find errors in it. It is preventive in nature and is completed in the verification phase. The common methods include feasibility review and code review. Software developers perform syntax check to test syntactical correctness. Dynamic testing refers to the analysis of the program in its executable form. This is performed by supplying valid entries and is validated against the expected results. It is a curative method and is performed during the validation phase.

Example: Software developers perform unit test to check for correctness in a module

Thus, static testing is done to check mainly the correctness and logic of the code whereas, dynamic testing is performed to check the response of the system for predefined inputs.

Since static testing is carried out during the initial stage, it is cost effective compared to dynamic testing, which is carried out once the entire software, a module, or unit is complete.

Static testing detects bugs at the earliest and hence the time required to fix them is less.

7.4.3 Manual versus Automated Testing

When the software is tested by people to find the bugs, it is called manual testing process. During this test, the tester acts as an end user and uses all the features of the software, and checks to ensure that they behave correctly. Performing a manual test for a login screen involves some of the following:

1. Checking whether username and password can be entered.

2. Implementing masking of password character.

3. Verifying whether the screen navigates to next page if valid login details are provided.

4. checking the error message displayed when incorrect login details are entered.

5. Checking the maximum number of characters that can be entered in the login and password fields. In automated testing, a software program, commonly referred to as 'a testing tool', runs the test software, provides proper inputs, and checks the output against the expected output. A tester writes the test case, and the automated testing tools run the test software according to the test case without any human intervention. At the end of the test, a detailed report is generated to indicate the result of any condition which is defined in the test case. Manual testing requires human intervention at every stage of the testing process right from writing test cases, providing the inputs, recording the output and analyzing the actual and expected output. There is always a chance for error in manual testing. It also requires more time to perform the test. Automated testing uses testing tools to perform the test. These tools need initial human intervention for supplying test cases. These tools are very powerful and reduce the time required for testing and are very efficient in finding out bugs in software. However, huge investment is required for using automated testing tools, since the tool has to be purchased from the vendor and some investment is also required for training. HP's Quality center is a popular automation tool used for testing and quality assurance.

Limitations of Manual Testing:

1. During the process of testing, the possibility of recurrence of a bug can actually have an impact on the time taken for testing the software.

2. Frequent changes in the user scenarios can lead to high maintenance costs in manual testing.

3. Repetitive tests reduce the cycle time for executing a test, and hence automation testing is opted for.

Depending on the limitations of time and cost factor, automation tests are preferred over manual tests.

Advantages of Automation Testing

1. It reduces the time consumed to perform repetitive tests.

2. It requires less human effort and less number of resources.

3. It generates a test report which provides information of the test execution unlike a manual test which is written and documented.

4. It helps in regression testing (Testing which is done to check whether the changes made in a module affects the working of the other existing modules) and also helps in re-running tests against new releases.

5. It helps in testing large sequences of data and transactions and also in randomly searching for errors in the software.

6. It helps in testing several simultaneous users at a time virtually and can also analyze the load generated for the program -- which cannot be done in manual testing.

7. It helps in testing web-based systems for performance reliability.

7.5 The V-Model of Software Testing

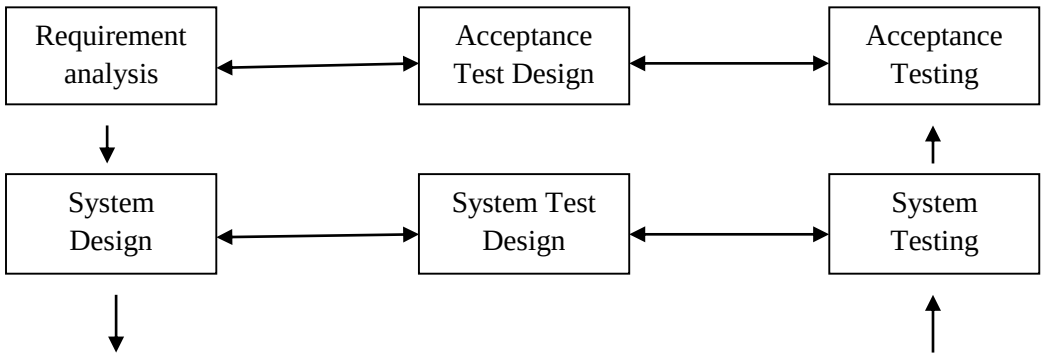
The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as Verification and Validation model.

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

V-Model - Design

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

The following illustration depicts the different phases in a V-Model of the SDLC.



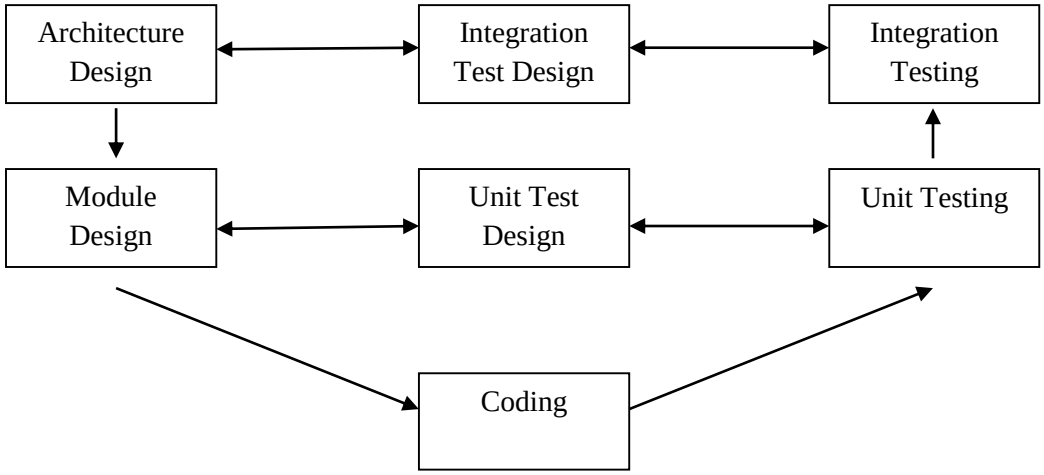


Figure: V-Model of Software Testing

7.5. 1 V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

1. ***Business Requirement Analysis***

This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

2. ***System Design***

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

3. ***Architectural Design***

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken.

The system design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

4. Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

5. Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

7.5.2 Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

7.5.3 V- Model ─ Application

V- Model application is almost the same as the waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly a disciplined domain.

The following pointers are some of the most suitable scenarios to use the V-Model application.

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

7.5.4 V-Model - Pros and Cons

The advantage of the V-Model method is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today's dynamic world, it becomes very expensive to make the change.

The advantages of the V-Model method are as follows −

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

The disadvantages of the V-Model method are as follows −

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change functionality.
- No working software is produced until late during the life cycle.

7.6 Levels of Software Testing

In general, there are four levels of testing: unit testing, integration testing, system testing and acceptance testing. The first three levels of testing activities are done by the testers and the last level of testing (acceptance) is done by the customer(s)/user(s). Each level has specific testing objectives. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level. For example, at the unit testing level, independent units are tested using functional and/or structural testing techniques. At the integration testing level, two or more units are combined and testing is carried out to test the integration related issues of various units. At the system testing level, the system is

tested as a whole and primarily functional testing techniques are used to test the system. Non-functional requirements like performance, reliability, usability, testability, etc. are also tested at this level. Load/stress testing is also performed at this level. The last level i.e. acceptance testing, is done by the customer(s)/user(s) for the purpose of accepting the final product.

There are many different testing levels which help to check behavior and performance for software testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. In SDLC models there are characterized phases such as requirement gathering, analysis, design, coding or execution, testing, and deployment. All these phases go through the process of software testing levels.

Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

7.6.1 Unit Testing:

Unit testing is a type of software testing where individual units or components of software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object. A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

Why Unit Testing?

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost defect fixing during system testing, integration and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

Here, are the key reasons to perform unit testing in software engineering:

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the testing code base and enables them to make changes quickly

3. Good unit tests serve as project documentation
4. Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.

How to do Unit Testing

In order **to do Unit Testing**, developers write a section of code to test a specific function in software application. Developers can also isolate this function to test more rigorously which reveals unnecessary dependencies between function being tested and other units so the dependencies can be eliminated. Developers generally use unit testing framework to develop automated test cases for unit testing.

7.6.2 Integration Testing

Integration testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

Why do Integration Testing?

Although each software module is unit tested, defects still exist for various reasons like

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the software modules work in unity
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues.

Example of Integration Test Case

Integration test case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules.** Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in unit testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Types of Integration testing:

1. Top Down Approach
2. Bottom Up Approach
3. Sandwich Approach

Stubs and Drivers

Stubs and Drivers are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitute for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.

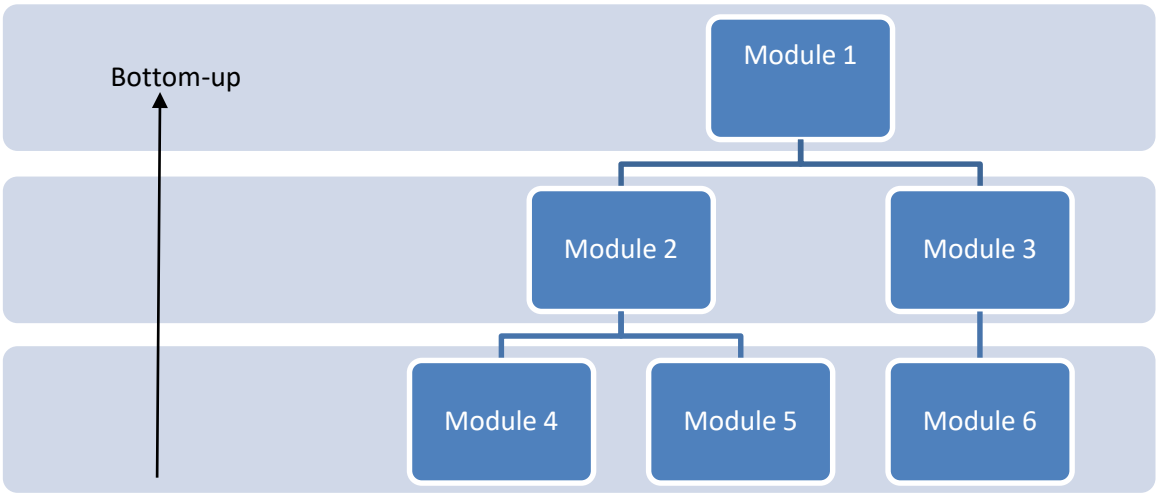
Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

1. Bottom-up Integration Testing

Bottom-up Integration Testing is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

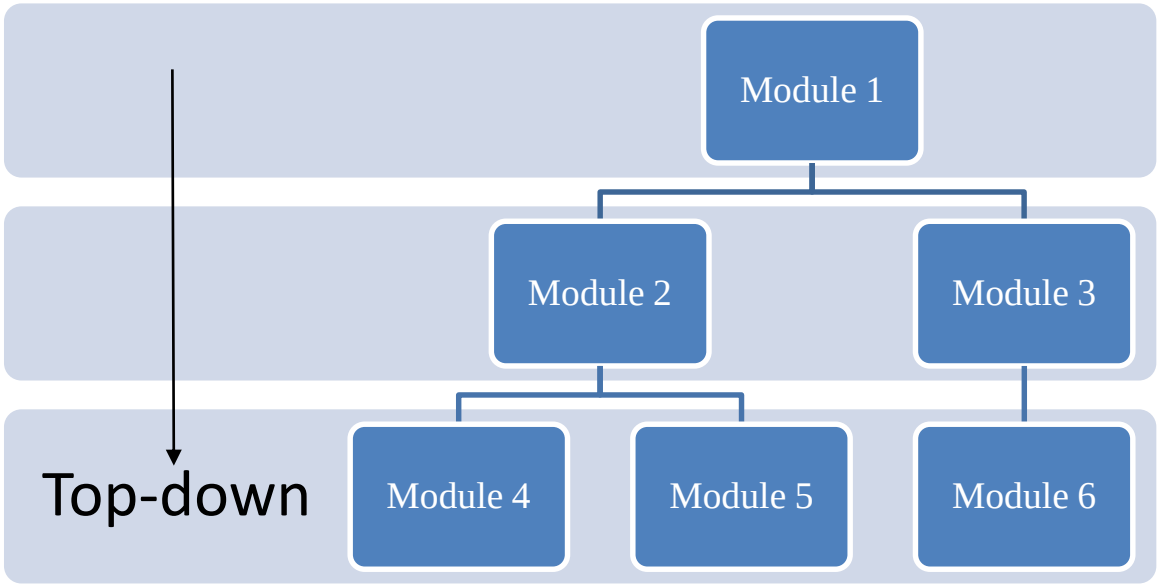
Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

2. Top-down Integration Testing

Top-down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

Diagrammatic Representation:



Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

3. Sandwich Testing

Sandwich Testing is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called **Hybrid Integration Testing.** It makes use of both stubs as well as drivers.

How to do Integration Testing?

The Integration test procedure irrespective of the Software testing strategies (discussed above):

1. Prepare the Integration Tests Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successful.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Smoke Testing

Smoke Testing comes into the picture at the time of receiving build software from the development team. The purpose of smoke testing is to determine whether the build software is testable or not. It is done at the time of "building software." This process is also known as "Day 0".

In other words, Smoke testing should be performed on each and every build without fail as it helps to find defects in early stages. Smoke test activity is the final step before the software build enters the system stage. Smoke tests must be performed on each build that is turned to testing. This applies to new development and major and minor releases of the system.

It is a time-saving process. It reduces testing time because testing is done only when the key features of the application are not working or if the key bugs are not fixed. The focus of Smoke Testing is on the workflow of the core and primary functions of the application. Before performing smoke testing, QA team must ensure the correct build version of the application under test. It is a simple process which takes a minimum time to test the stability of the application.

Smoke tests can minimize test effort, and can improve the quality of the application. Smoke testing can be done either manually or by automation depending on the client and the organization. Testing the basic & critical feature of an application before doing one round of deep, rigorous testing (before checking all possible positive and negative values) is known as smoke testing.

In the smoke testing, we only focus on the positive flow of the application and enter only valid data, not the invalid data. In smoke testing, we verify every build is testable or not; hence it is also known as **Build Verification Testing.**

When do we do smoke testing

Smoke Testing is done whenever the new functionalities of software are developed and integrated with existing build that is deployed in QA/staging environment. It ensures that all critical functionalities are working correctly or not. In this testing method, the development team deploys the build in QA. The subsets of test cases are taken, and then testers run test cases on the build. The QA team tests the application against the critical functionalities. These series of test cases are designed to expose errors that are in build. If these tests are passed, QA team continues with functional testing.

Any failure indicates a need to handle the system back to the development team. Whenever there is a change in the build, we perform Smoke Testing to ensure the stability.

Example: -New registration button is added in the login window and build is deployed with the new code. We perform smoke testing on a new build.

Who will do Smoke Testing

After releasing the build to QA environment, Smoke Testing is performed by QA engineers/QA lead. Whenever there is a new build, QA team determines the major functionality in the application to perform smoke testing. QA team checks for showstoppers in the application that is under testing.

Testing done in a development environment on the code to ensure the correctness of the application before releasing build to QA, this is known as Sanity testing. It is usually narrow and deep testing. It is a process which verifies that the application under development meets its basic functional requirements.

Why do we do smoke testing?

Smoke testing plays an important role in software development as it ensures the correctness of the system in initial stages. By this, we can save test effort. As a result, smoke tests bring the system to a good state. Once we complete smoke testing then only we start functional testing.

- All the show stoppers in the build will get identified by performing smoke testing.
- Smoke testing is done after the build is released to QA. With the help of smoke testing, most of the defects are identified at initial stages of software development.
- With smoke testing, we simplify the detection and correction of major defects.
- By smoke testing, QA team can find defects to the application functionality that may have surfaced by the new code.
- Smoke testing finds the major severity defects.

Example 1: Logging window: Able to move to next window with valid username and password on clicking submit button.

Example 2: User unable to sign out from the webpage.

Smoke testing cycle

Below flow chart shows how Smoke Testing is executed. Once the build is deployed in QA and, smoke tests are passed we proceed for functional testing. If the smoke test fails, we exit testing until the issue in the build is fixed.

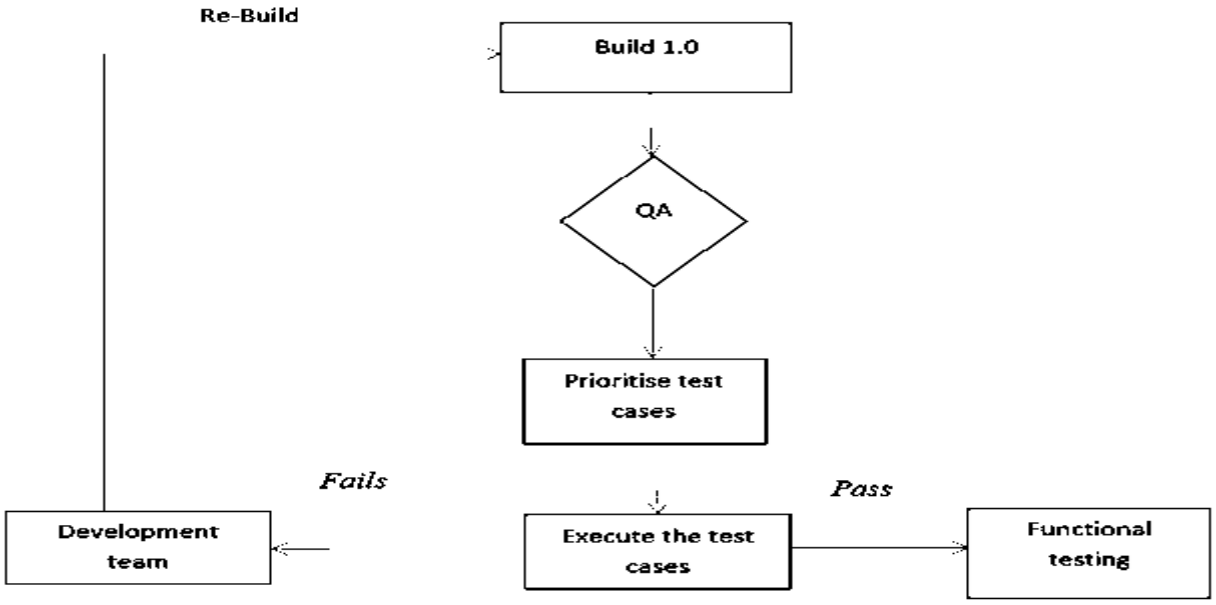


Figure: Smoke test cycle

Advantages of Smoke testing

Here are few advantages listed for Smoke Testing.

- Easy to perform testing
- Defects will be identified in early stages.
- Improves the quality of the system
- Reduces the risk
- Progress is easier to access.
- Saves test effort and time
- Easy to detect critical errors and correction of errors.
- It runs quickly
- Minimises integration risks

What happens if we don't do Smoke testing

If we don't perform smoke testing in early stages, defects may be encountered in later stages where it can be cost effective. And the defect found in later stages can be show stoppers where it may affect the release of deliverables.

7.6.3 System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.

System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

7.6.4 Acceptance Testing

Acceptance testing, a testing technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.

Like all other testing steps, validation tries to uncover errors, but the focus is at the requirements level-on things that will be immediately apparent to the end-user.

7.7 System Testing

System Testing is actually a series of different tests to whose primary purpose is to fully exercise the computer-based system. There are different testing's are performed under system testing:

7.7.1 Recovery Testing

Recovery Testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. The purpose of Recovery Testing is to determine whether software operations can be continued after disaster or integrity loss. Recovery testing involves reverting back software to the point where integrity was known and reprocessing transactions to the failure point.

For example:

When an application is receiving data from the network, unplug the connecting cable.

- After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection was broken.
- Restart the system while a browser has a definite number of sessions open and check whether the browser is able to recover all of them or not

In Software Engineering, Recoverability Testing is a type of Non-functional testing. (Non- functional testing refers to aspects of the software that may not be related to a specific function or user action such as scalability or security.)

The time taken to recover depends upon:

- The number of restart points
- A volume of the applications
- Training and skills of people conducting recovery activities and tools available for recovery.

When there are a number of failures then instead of taking care of all failures, the recovery testing should be done in a structured fashion which means recovery testing should be carried out for one segment and then another.

It is done by professional testers. Before recovery testing, adequate backup data is kept in secure locations. This is done to ensure that the operation can be continued even after a disaster

7.7.2 Security Testing

Security Testing is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders. The purpose of Security Tests is to identify all possible loopholes and weaknesses of the software system which might result in a loss of information, revenue, repute at the hands of the employees or outsiders of the Organization.

Why Security Testing is Important?

The main goal of **Security Testing** is to identify the threats in the system and measure its potential vulnerabilities, so the threats can be encountered and the system does not stop functioning or cannot be exploited. It also helps in detecting all possible security risks in the system and helps developers to fix the problems through coding.

Key Areas in Security Testing

While performing the security testing on the web application, we need to concentrate on the following areas to test the application:

Types of Security Testing:

There are seven main types of security testing as per Open Source Security Testing methodology manual. They are explained as follows:

- ***Vulnerability Scanning****:* This is done through automated software to scan a system against known vulnerability signatures.
- ***Security Scanning:*** It involves identifying network and system weaknesses, and later provides solutions for reducing these risks. This scanning can be performed for both Manual and Automated scanning.
- ***Penetration testing****:* This kind of testing simulates an attack from a malicious hacker. This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.
- ***Risk Assessment:*** This testing involves analysis of security risks observed in the organization. Risks are classified as Low, Medium and High. This testing recommends controls and measures to reduce the risk.

- ***Security Auditing:*** This is an internal inspection of Applications and Operating systems for security flaws. An audit can also be done via line by line inspection of code
- ***Ethical hacking:*** It's hacking an Organization Software system. Unlike malicious hackers, who steal for their own gains, the intent is to expose security flaws in the system.
- ***Posture Assessment:*** This combines Security scanning, Ethical Hacking and Risk Assessments to show an overall security posture of an organization.

How we perform security testing

The security testing is needed to be done in the initial stages of the Software development life cycle because if we perform security testing after the software execution stage and the deployment stage of the SDLC, it will cost us more.

Now let us understand how we perform security testing parallel in each stage of the software development life cycle (SDLC).

Step1

SDLC: Requirement stage

Security Procedures: In the requirement phase of SDLC, we will do the security analysis of the business needs and also verify that which cases are manipulative and waste.

Step2

SDLC: Design stage

Security Procedures: In the design phase of SDLC, we will do the **security testing for risk** exploration of the design and also embraces the security tests at the development of the test plan.

Step3

SDLC: Development or coding stage

Security Procedures: In the coding phase of SDLC, we will perform the white box testing along with static and dynamic testing.

Step4

SDLC: Testing (functional testing, integration testing, system testing) stage

Security Procedures: In the testing phase of SDLC, we will do one round of **vulnerability scanning** along with black-box testing.

Step 5

SDLC: Implementation stage

Security Procedures: In the implementation phase of SDLC, we will perform **vulnerability scanning** again and also perform one round of **penetration testing.**

Step 6

SDLC: Maintenance stage

Security Procedures: In the Maintenance phase of SDLC, we will do the **impact analysis** of impact areas.

And the **test plan** should contain the following:

- The test data should be linked to security testing.
- For security testing, we need the test tools.
- With the help of various security tools, we can analyze several test outputs.
- Write the test scenarios or test cases that rely on security purposes.

Example Test Scenarios for Security Testing:

- A password should be in encrypted format
- Application or System should not allow invalid users
- Check cookies and session time for application
- For financial sites, the Browser back button should not work.

Why security testing is essential for web applications

At present, web applications are growing day by day, and most of the web application is at risk. Here we are going to discuss some common weaknesses of the web application.

- Client-side attacks
- Authentication
- Authorization
- Command execution
- Logical attacks
- Information disclosure

System software security

In this, we will evaluate the vulnerabilities of the application based on different software such as **Operating system, Database system** etc.

Network security

In this, we will check the weakness of the network structure, such as **policies and resources.**

Server-side application security

We will do the server-side application security to ensure that the server encryption and its tools are sufficient to protect the software from any disturbance.

Client-side application security

In this, we will make sure that any intruders cannot operate on any browser or any tool which is used by customers.

7.7.3 Stress Testing

Stress testing includes testing the behavior of software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

7.7.4 Performance Testing

Performance Testing is a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application. It is a subset of performance engineering and also known as "Perf Testing".

The focus of Performance Testing is checking a software programs

- Speed – Determines whether the application responds quickly
- Scalability – Determines maximum user load the software application can handle.
- Stability – Determines if the application is stable under varying loads

Why do Performance Testing?

Features and Functionality supported by a software system is not the only concern. A software application's performance like its response time, reliability, resource usage and scalability do matter. The goal of Performance Testing is not to find bugs but to eliminate performance bottlenecks.

Performance Testing is done to provide stakeholders with information about their application regarding speed, stability, and scalability. More importantly, Performance Testing uncovers what needs to be improved before the product goes to market. Without Performance Testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability.

Types of Performance Testing

- ***Load testing –*** checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

- ***Stress testing* –** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.
- ***Endurance testing* –** is done to make sure the software can handle the expected load over a long period of time.
- ***Spike testing* –** tests the software's reaction to sudden large spikes in the load generated by users.
- ***Volume testing*** – Under Volume Testing large no. of. Data is populated in a database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.
- ***Scalability testing*** – The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity addition to your software system.

Performance Testing Process:

The methodology adopted for performance testing can vary widely but the objective for performance tests remain the same. It can help demonstrate that your software system meets certain pre-defined performance criteria. Or it can help compare the performance of two software systems. It can also help identify parts of your software system which degrade its performance.

Below is a generic process on how to perform performance testing.

1. ***Identify your testing environment* –** Know your physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before you begin the testing process. It will help testers create more efficient tests. It will also help identify possible challenges that testers may encounter during the performance testing procedures.

2. ***Identify the performance acceptance criteria*** **–** This includes goals and constraints for throughput, response times and resource allocation. It is also necessary to identify project success criteria outside of these goals and constraints. Testers should be empowered to set performance criteria and goals because often the project specifications will not include a wide enough variety of performance benchmarks. Sometimes there may be none at all. When possible finding a similar application to compare to is a good way to set performance goals.
3. ***Plan & design performance tests*** **–** Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
4. ***Configuring the test environment*** **–** Prepare the testing environment before execution. Also, arrange tools and other resources.
5. ***Implement test design*** **–** Create the performance tests according to your test design.
6. ***Run the tests*** **–** Execute and monitor the tests.
7. ***Analyze, tune and retest*** – Consolidate, analyze and share test results. Then fine tune and test again to see if there is an improvement or decrease in performance. Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU. Then you may have the consider option of increasing CPU power.

Example Performance Test Cases

- Verify response time is not more than 4 secs when 1000 users access the website simultaneously.
- Verify response time of the Application Under Load is within an acceptable range when the network connectivity is slow
- Check the maximum number of users that the application can handle before it crashes.
- Check database execution time when 500 records are read/ written simultaneously.
- Check CPU and memory usage of the application and the database server under peak load conditions

- Verify response time of the application under low, normal, moderate and heavy load conditions.

7.8 Acceptance Testing

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.

Acceptance Criteria

Acceptance criteria are defined on the basis of the following attributes

- Functional Correctness and Completeness
- Data Integrity
- Data Conversion
- Usability
- Performance
- Timeliness
- Confidentiality and Availability
- Installability and Upgradability
- Scalability
- Documentation

Acceptance testing types:

There are various forms of acceptance testing:

1. Alpha Testing
2. Beta Testing
3. Gamma Testing

7.8.1 Alpha Testing

Alpha Testing is a type of acceptance testing; performed to identify all possible issues and bugs before releasing the final product to the end users. Alpha testing is carried out by the testers who are internal employees of the organization. The main goal is to identify the tasks that a typical user might perform and test them.

To put it as simple as possible, this kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing. The main focus of alpha testing is to simulate real users by using a black box and white box techniques.

7.8.2 Beta Testing

Beta Testing is performed by "real users" of the software application in "real environment" and it can be considered as a form of external User Acceptance Testing. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in customer's environment.

Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.

Difference between alpha and beta testing

- Alpha Testing is performed by the Testers within the organization whereas Beta Testing is performed by the end users.
- Alpha Testing is performed at Developer's site whereas Beta Testing is performed at Client's location.
- Reliability and Security testing are not performed in-depth in Alpha Testing while Reliability, Security and Robustness are checked during Beta Testing.
- Alpha Testing involves both White box and Black box testing whereas Beta Testing mainly involves Black box testing.

- Alpha Testing requires testing environment while Beta Testing doesn't require testing environment.
- Alpha Testing requires long execution cycle whereas Beta Testing requires only few weeks of execution.
- Critical issues and bugs are addressed and fixed immediately in Alpha Testing whereas issues and bugs are collected from the end users and further implemented in Beta Testing.

Advantages of Alpha Testing:

- Provides better view about the reliability of the software at an early stage
- Helps simulate real time user behavior and environment.
- Detect many showstopper or serious errors
- Ability to provide early detection of errors with respect to design and functionality

Advantages of Beta Testing

- Reduces product failure risk via customer validation.
- Beta Testing allows a company to test post-launch infrastructure.
- Improves product quality via customer feedback
- Cost effective compared to similar data gathering methods
- Creates goodwill with customers and increases customer satisfaction

Disadvantages of Alpha Testing:

- In depth, functionality cannot be tested as software is still under development stage Sometimes developers and testers are dissatisfied with the results of alpha testing

Disadvantages of Beta Testing

- Test Management is an issue. As compared to other testing types which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you seldom have control.
- Finding the right beta users and maintaining their participation could be a challenge

7.8.3 Gamma Testing:

Gamma Testing is done when software is ready for release with specified requirements, this testing done directly by skipping all the in-house testing activities. The software is almost ready for final release. No feature development or enhancement of the software is undertaken and tightly scoped bug fixes are the only code.

Gamma check is performed when the application is ready for release to the specified requirements and this check is performed directly without going through all the testing activities at home.

Gamma testing is the third level of testing, generally for safety. Unfortunately, Gamma testing is becoming a thing of the past, killed off by decreased time cycles, competitive pressure, and the myopic focus on quarterly profits.

7.9 Software Test Report (STR)

Software Test Report is a document which contains a summary of all test activities and final test results of a testing project. Test report is an assessment of how well the testing is performed. Based on the test report, stakeholders can evaluate the quality of the tested product and make a decision on the software release.

For example, if the test report informs that there are many defects remaining in the product, stakeholders can delay the release until all the defects are fixed.

What does a test report contain?

1. ***Project Information***

All information of the project such as the project name, product name, and version should be described in the test report.

2. ***Test Objective***

Test Report should include the objective of each round of testing, such as Unit Test, Performance Test, System Test …Etc.

3. ***Test Summary***

This section includes the summary of testing activity in general. Information detailed here includes

- The number of test cases executed
- The numbers of test cases pass
- The numbers of test cases fail
- Pass percentage
- Fail percentage
- Comments

4. Defect

One of the most important information in Test Report is defect. The report should contain following information

- Total number of bugs
- Status of bugs (open, closed, responding)
- Number of bugs open, resolved, closed
- Breakdown by severity and priority

Like test summary, you can include some simple metrics like defect density, % of fixed defects.

To solve that problem, a good Test Report should be:

- ***Detail****:* You should provide a detailed description of the testing activity, show which testing you have performed. Do not put the abstract information into the report, because the reader will not understand what you said.
- ***Clear:*** All information in the test report should be **short** and **clearly** understandable.
- ***Standard*****:** The Test Report should follow the **standard** template. It is easy for stakeholder to review and ensure the **consistency** between test reports in many projects.
- ***Specific*****:** Do not write an essay about the project activity. Describe and summarize the test result specification and focus on the main point.

For example, to correct the above Test Report, the tester should provide more information such as:

- Project information
- Test cycle: (System Test, Integration Test…etc.)
- Which functions have already tested (% Test cycles executed, % Test cycles passed or fail)
- Defect report (Defect description, Priority or status)

7.10 Software Testing Tools

The most important effort-consuming task in software testing is to design the test cases. The execution of these test cases may not require much time and resources. Hence, the designing part is more significant than the execution part. Both parts are normally handled manually. Do we really need a tool? If yes, where and when can we use it – in the first part (designing of test cases) or second part (execution of test cases) or both? Software testing tools may be used to reduce the time of testing and to make testing as easy and pleasant as possible. Automated testing may be carried out without human involvement. This may help us in the areas where a similar dataset is to be given as input to the program again and again. A tool may undertake repeated testing, unattended (and without human intervention), during nights or on weekends. Many non-functional requirements may be tested with the help of a tool. We want to test the performance of software under load, which may require many computers, manpower and other resources. A tool may simulate multiple users on one computer and also a situation when many users are accessing a database simultaneously.

There are two broad categories of software testing tools i.e. static and dynamic. Most of the tools fall clearly into one of these categories but there are a few exceptions like mutation analysis system which falls in more than one category. A wide variety of tools are available with different scope and quality and they assist us in many ways.

7.10.1 Static Software Testing Tools

Static software testing tools are those that perform analysis of the programs without executing them at all. They may also find the source code which will be hard to test and maintain. As we 380 Software testing all know, static testing is about prevention and dynamic testing is about cure. We should use both the tools but prevention is always better than cure. These tools will find more bugs as compared to dynamic testing tools (where we execute the program). There are many areas for which effective static

testing tools are available, and they have shown their results for the improvement of the quality of the software.

(i) Complexity analysis tools

Complexity of a program plays a very important role while determining its quality. A popular measure of complexity is the cyclomatic complexity as discussed in chapter 4. This gives us the idea about the number of independent paths in the program and is dependent upon the number of decisions in the program. A higher value of cyclomatic complexity may indicate poor design and risky implementation. This may also be applied at the module level, and higher cyclomatic complexity value modules may either be redesigned or may be tested very thoroughly. There are other complexity measures also which are used in practice like Halstead software size measures, knot complexity measure, etc. Tools are available which are based on any of the complexity measures. These tools may take the program as an input, process it and produce a complexity value as output. This value may be an indicator of the quality of design and implementation.

(ii) Syntax and semantic analysis tools

These tools find syntax and semantic errors. Although the compiler may detect all syntax errors during compilation, early detection of such errors may help to minimize other associated errors. Semantic errors are very significant and compilers are helpless in finding such errors. There are tools in the market that may analyze the program and find errors. Non-declaration of a variable, double declaration of a variable, 'divide by zero' issue, unspecified inputs and non-initialization of a variable are some of the issues which may be detected by semantic analysis tools. These tools are language dependent and may parse the source code, maintain a list of errors and provide implementation information. The parser may find semantic errors as well as make an inference as to what is syntactically correct.

(iii) Flow graph generator tools

These tools are language dependent and take the program as an input and convert it to its flow graph. The flow graph may be used for many purposes like complexity calculation, paths identification, generation of definition use paths, program slicing, etc. These tools assist us to understand the risky and poorly designed areas of the source code.

(iv) Code comprehension tools

These tools may help us to understand unfamiliar source code. They may also identify dead source code, duplicate source code and areas that may require special attention and should be reviewed seriously.

(v) Code inspectors

Source code inspectors do the simple job of enforcing standards in a uniform way for many programs. They inspect the programs and force us to implement the guidelines of good programming practices. Although they are language dependent, most of the guidelines of good programming practices are similar in many languages. These tools Software Testing Activities 381 are simple and may find many critical and weak areas of the program. They may also suggest possible changes in the source code for improvement.

7.10.2 Dynamic Software Testing Tools

Dynamic software testing tools select test cases and execute the program to get the results. They also analyze the results and find reasons for failures (if any) of the program. They will be used after the implementation of the program and may also test non-functional requirements like efficiency, performance, reliability, etc.

(i) Coverage analysis tools

These tools are used to find the level of coverage of the program after executing the selected test cases. They give us an idea about the effectiveness of the selected test cases. They highlight the unexecuted portion of the source code and force us to design special test cases for that portion of the source code. There are many levels of coverage like statement coverage, branch coverage, condition coverage, multiple condition coverage, path coverage, etc. We may like to ensure that at least every statement must be executed once and every outcome of the branch statement must be executed once. This minimum level of coverage may be shown by a tool after executing an appropriate set of test cases. There are tools available for checking statement coverage, branch coverage, condition coverage, multiple conditions coverage and path coverage. The profiler displays the number of times each statement is executed. We may study the output to know which portion of the source code is not executed. We may design test cases for those portions of the source code in order to achieve the desired level of coverage. Some tools are also available to check whether the source code is as per standards or

not and also generate a number of commented lines, non-commented lines, local variables, global variables, duplicate declaration of variables, etc. Some tools check the portability of the source code. A source code is not portable if some operating system dependent features are used. Some tools are Automated QA's time, Parasoft's Insure++ and Telelogic's Logicscope.

(ii) Performance testing tools

We may like to test the performance of the software under stress / load. For example, if we are testing result management software, we may observe the performance when 10 users are entering the data and also when 100 users are entering the data simultaneously. Similarly, we may like to test a website with 10 users, 100 users, 1000 users, etc. working simultaneously. This may require huge resources and sometimes, it may not be possible to create such real life environment for testing in the company. A tool may help us to simulate such situations and test these situations in various stress conditions. This is the most popular area for the usage of any tool and many popular tools are available in the market. These tools simulate multiple users on a single computer. We may also see the response time for a database when 10 users access the database, when 100 users access the database and when 1000 users access the data base simultaneously. Will the response time be 10 seconds or 100 seconds or even 1000 seconds? No user may like to tolerate the response time in minutes. Performance testing includes load 382 Software Testing and stress testing. Some of the popular tools are Mercury Interactive's Load Runner, Apache's J Meter, Segue Software's Silk Performer, IBM Rational's Performance Tester, Comuware's QALOAD and AutoTester's AutoController.

(iii) Functional / Regression Testing Tools

These tools are used to test the software on the basis of its functionality without considering the implementation details. They may also generate test cases automatically and execute them without human intervention. Many combinations of inputs may be considered for generating test cases automatically and these test cases may be executed, thus, relieving us from repeated testing activities. Some of the popular available tools are IBM Rational's Robot, Mercury Interactive's Win Runner, Comuware's QA Centre and Segue Software's Silktest.

7.11 CHECK YOUR PROGRESS

1. _____ is a process of executing a program with the intent of finding errors.
2. Test artifacts are also known as _____.
3. Bottom-up and top-down are _____ testing techniques.
4. In_____, the test engineer implements the same test cases on a modified build.
5. STLC stands for _____ .
6. Functionality of software is tested by _____.
7. Site for Alpha testing is _____ .
8. _____ includes detecting and modifying code errors in a software program.
9. _____ is checking the product with respect to customer's expectation.
10. Testing of software with actual data and in the actual environment is called _____ .

7.12 SUMMARY

This chapter establishes the basics of software testing. It also presents the definitions of error, bug, fault, inspection, successful tester and the basic principle of software testing. It explains how testing evolved from mere debugging to defect prevention technique. It offers a detailed exposition of the process of creating test policy, test strategy, and test plan. It also gives an in depth understanding of 'Black Box testing' and 'white box testing'. It offers an elaborate discussion of 'V model' of software Validation. We have also seen activities related to verification and validation. Understanding of these activities is crucial to derive a good product.

We have seen advantages and disadvantages of types of testing such as regression testing, stress testing, security testing, performance testing, smoke testing, acceptance testing etc. We have also explain various level of testing such as unit, integration, system, acceptance testing. Integration testing further divided into three types: bottom-up, top-down and sandwich testing. Acceptance testing is a planned activity and must consider critically of the system and cost benefit analysis. We have studied various types of acceptance testing such as alpha, beta and gamma. Later, we have discussed the introduction of

software testing tools like static testing tool, dynamic testing tool. This chapter concludes with skills required by a good tester and challenges faced by a tester.

7.13 SELF ASSESSMENT TEST

1. Define the following:
   (a) Stress testing
   (b) Verification
   (c) Regression testing
   (d) Security testing
   (e) Smoke testing
2. What is the v-model of software testing? Explain different stages with suitable example
3. (a) Differentiate between testing and debugging.
   (b) Differentiate between inspection and testing.
4. What is acceptance testing? Explain its type.
5. Compare black box testing with white box testing.
6. Explain the different level of testing with example.

7.14 ANSWER TO CHECK YOUR PROGRESS

1. Test Deliverables
2. Testing
3. Integration
4. Regression testing
5. Software testing life cycle
6. Black Box testing
7. Software company
8. Debugging

9. Validation
10. Beta testing

7.15 REFERENCES/SUGGESTED READINGS

1. M G LIMAYE, “SoftwareTesting”, Mc Graw Hill Education
2. K.K. Aggarwal & Yogesh Singh, “Software Engineering”, New age international publishers

SUBJECT: Software Engineering

COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 8	

Computer Aided Software Engineering

8.1 Learning objectives

This lesson emphasis on importance of CASE tools in SDLC , various types of CASE tools , case tool environment , and its components , architecture of good CASE tool . CASE tools related to Phase related activates and non-phase activates of SDLC are also discussed individually.

8.2 Introduction

"Software" as a term is most meaningful when you use it to distinguish from "hardware" or even non-automated information processing by humans. However, it is also generic, like "program", and so it can refer to any code that runs on digital computer. ... A "tool" is software that doesn't provide end-user value on it's own. System software tools provide the platform for running application or application software. It brings together the capabilities of the computer. Tasks that benefit end users fall under the domain of application software. ERP and supply chain management includes applications for services and manufacturing. The most basic tools are a source code editor and a compiler or interpreter, which are used ubiquitously and continuously. Other tools are used more or less depending on the language, development methodology, and individual engineer, often used for a discrete task, like a debugger or profiler. Whenever a new system is installed, the implementation integrates a number of related and different tasks. The process has to be efficiently organized and it is for this very reason that CASE tools are developed. With the help of CASE, the installation process can be automated and coordinated within the developed and adopted system life cycle.

CASE tools are the software engineering tools that permit collaborative software development and maintenance. Almost all the phases of the software development life cycle are supported by them such as analysis; design, etc., including umbrella activities such as project management, configuration management etc. In general, standard software development methods such as Jackson Structure programming or structured system analysis and design method are also supported by CASE tools. CASE tools may support the following development steps for developing data base application:

- Creation of data flow and entity models
- Establishing a relationship between requirements and models
- Development of top-level design
- Development of functional and process description
- Development of test cases.

Computer Aided Software Engineering(CASE) is software that supports one or more software engineering activities within a software development process, and is gradually becoming popular for the development of software as they are improving in the capabilities and functionality and are proving to be beneficial for the development of quality software.

A CASE (Computer Aided Software Engineering) tool is a generic term used to denote any form of automated support for software engineering. In a more restrictive sense, a CASE tool means any tool used to automate some activity associated with software development. Various tools are incorporated in CASE and are called CASE tools, which are used to support different stages and milestones in a software development life cycle.

Basically we have two categories of CASE tools based upon tasks associated with software development life cycle :-

a) phase related tasks such as specification, structured analysis, design, coding, testing, etc.;

b) non-phase activities such as project management and configuration management.

Computer aided software engineering (CASE) is the implementation of computer facilitated tools and methods in software development. CASE is used to ensure a high-quality and defect-free software. CASE ensures a check-pointed and disciplined approach and helps designers, developers, testers, managers and others to see the project milestones during development.

The essential idea of CASE tools is that in-built programs can help to analyze developing systems in order to enhance quality and provide better outcomes. Throughout the 1990, CASE tool became part of the software , and big companies like IBM were using these kinds of tools to help create software.

The central component of any CASE tool is the CASE repository, otherwise known as the information repository or data dictionary. The CASE repository stores the diagrams and other project information, such as screen and report designs, and it keeps track of how the diagrams fit together.

Generally, Upper CASE is a tool for high level view of software development whereas lower CASE is mostly being used as a tool at the programming and testing phase. Evaluation of CASE tools has been conducted on several academic studies defines a method for evaluating and selecting CASE tools.

8.3 CASE environment

CASE is the use of computer-based support in the software development process; a **CASE** tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process; a **CASE environment** is a collection of **CASE** tools and other components together with an integration ... Although individual CASE tools are useful, the true power

of a tool set can be realized only when these set of tools are integrated into a common framework or environment. CASE tools are characterized by the stage or stages of software development life cycle on which they focus. Since different tools covering different stages share common information, it is required that they integrate through some central repository to have a consistent view of information associated with the software development artifacts. This central repository is usually a data dictionary containing the definition of all composite and elementary Version 2 CSE IIT, Kharagpur data items. Through the central repository all the CASE tools in a CASE environment share common information among themselves. Thus a CASE environment facilities the automation of the step-by-step methodologies for software development. A schematic representation of a CASE environment is shown in Figure 8.1.

CASE tools square measure characterized by the stage or stages of package development life cycle that they focus on.

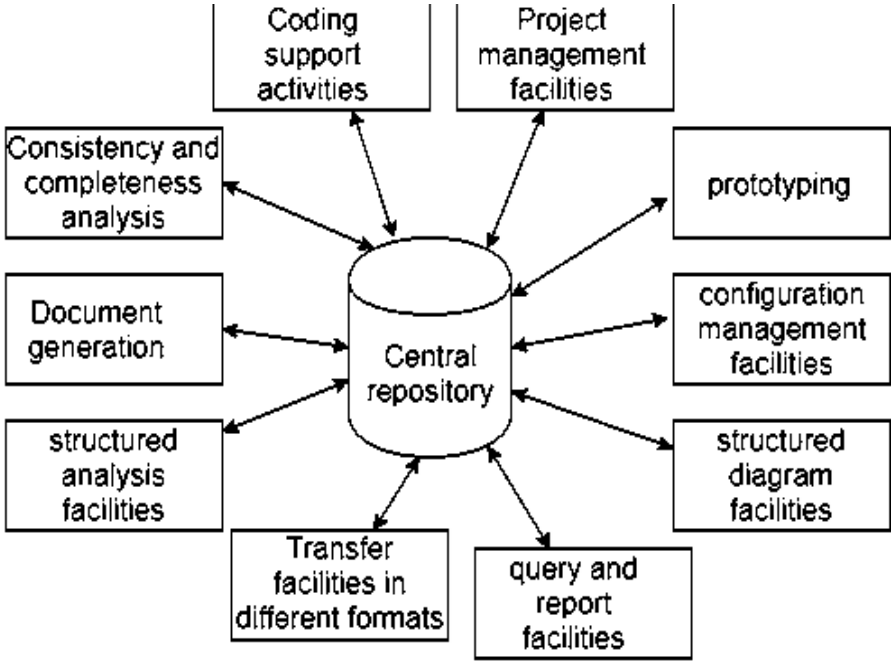


Figure 8.1 ACASE Environment

Since totally different tools covering different stages share common data, it's needed that they integrate through some central repository to possess an even read of data related to the package development

artifacts. This central repository is sometimes information lexicon containing the definition of all composite and elementary data things.

Through the central repository, all the CASE tools in a very CASE setting share common data among themselves. therefore a CASE setting facilities the automation of the step-wise methodologies for package development.

8.3.1 CASE environment vs programming environment

It should be noted that CASE environment is different from programming environment.

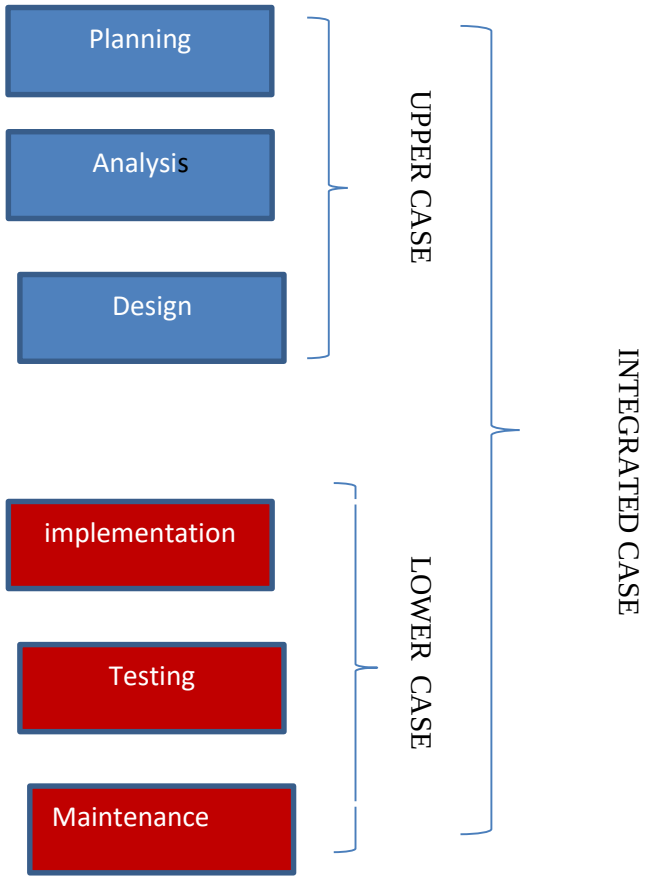
A CASE environment facilitates the automation of the step-by-step methodologies for software development. In contrast to a CASE environment, a programming environment is an integrated collection of tools to support only the coding phase of software development.

A CASE environment facilitates the automation of the in small stages methodologies for package development. In distinction to a CASE environment, a programming environment is an Associate in a Nursing integrated assortment of tools to support solely the cryptography part of package development.

8.4 Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

- **Central Repository** - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.



- **Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.
- **Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation. 3.The integrated CASE tool supports all phases of SDLC and provides the functionality of both upper-CASE and lower-CASE in one tool. It is known as I-CASE and also supports analysis, design and coding phases.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

The phases that are supported by the Lower and upper CASE tools are shown in the Waterfall model as under(see Figure 8.2):

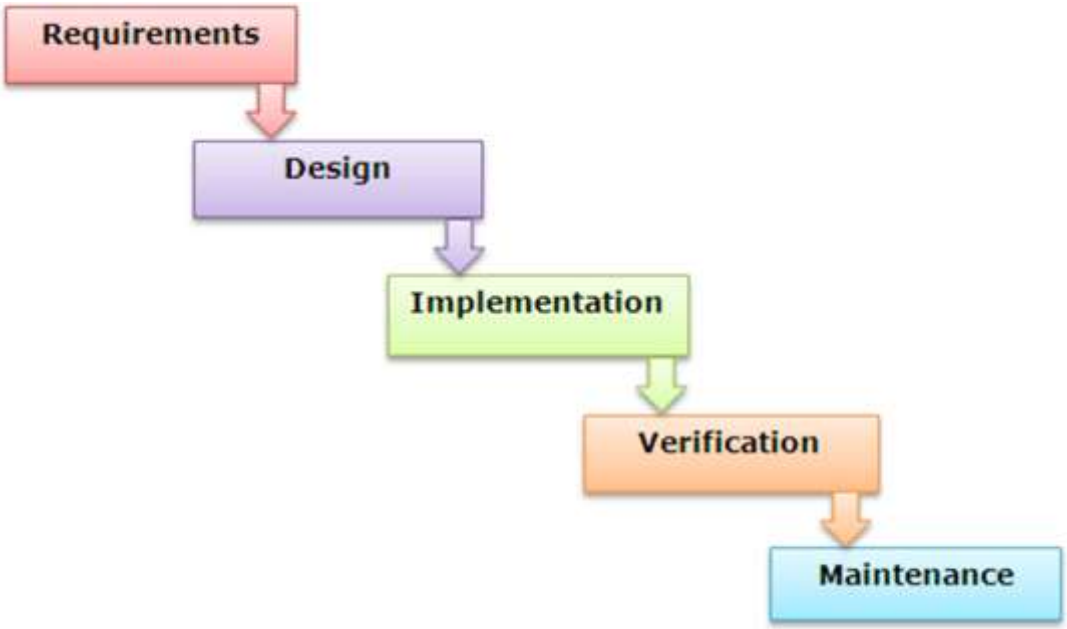


Figure 8.2

Positioning of CASE tools in a Software Application development:

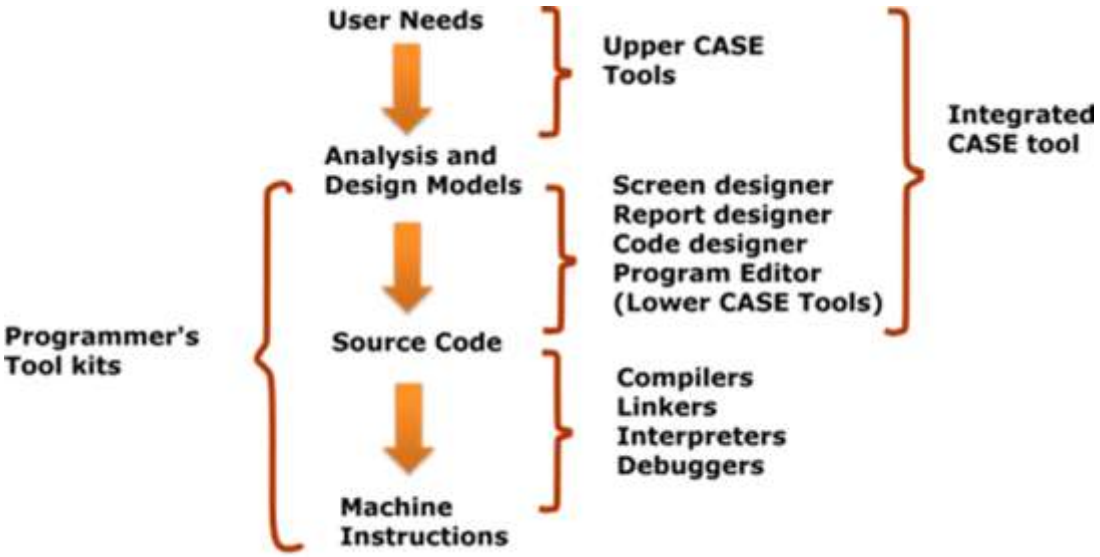


Figure 8.3 CASE Tool and application development

8.5 Architecture of a CASE environment

Following figure 8.4 shows **Architecture of a CASE environment**

Elements of CASE atmosphere are

- computer program,
- toolset,
- object management system (OMS),
- repository.

These elements are dynamic in nature.

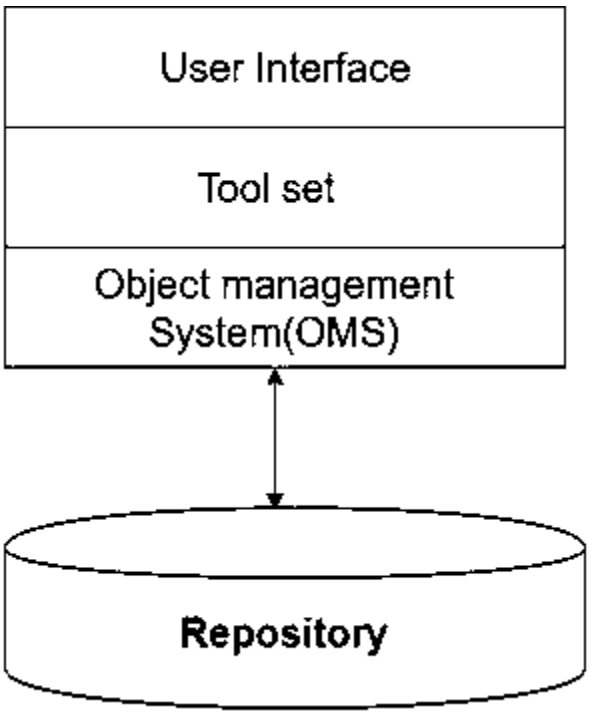


figure 8.4 Architecture of CASE Environment

User Interface:

the user interface provides a regular framework for accessing the various tools so creating it easier for the users to act with the different tools and reducing the overhead of learning however the different tools are used.

Object Management System (OMS) and Repository:

Different case tools represent the product as a group of entities like specification, design, text data,

project arrange, etc. the thing management system maps these logical entities such into the underlying storage management system (repository).

8.6 Types of CASE Tools:

i) Diagramming Tools:

It helps in diagrammatic and graphical representations of the data and system processes. It represents system elements, control flow and data flow among different software components and system structure in a pictorial form.
For example, Flow Chart Maker tool for making state-of-the-art flowcharts.

Computer Display and Report Generators:

It helps in understanding the data requirements and the relationships involved.

ii) Analysis Tools:

It focuses on inconsistent, incorrect specifications involved in the diagram and data flow. It helps in collecting requirements, automatically check for any irregularity, imprecision in the diagrams, data redundancies or erroneous omissions.
For example,

(a) Accept 360, Accompa, CaseComplete for requirement analysis.
(b) Visible Analyst for total analysis.

iv) Central Repository:

It provides the single point of storage for data diagrams, reports and documents related to project management.

v) Documentation Generators:

It helps in generating user and technical documentation as per standards. It creates documents for technical users and end users.
For example, Doxygen, DrExplain, Adobe RoboHelp for documentation**.**

vi) Code Generators:

It aids in the auto generation of code, including definitions, with the help of the designs, documents and diagrams.

8.7 Advantages and Disadvantages of CASE Tools

a) Advantages

CASE tools were created to improve the productivity and quality of database development. The advantages of using them are:

- Produce system with a longer effective operational life.
- Produces system that more closely meet user needs and requirements.
- Produces system with excellent documentation.
- Produce more flexible system.

b) Disadvantages

- Produce initial system that is more expensive to build and maintain
- Require more extensive and accurate definitions of user needs and requirements.
- May be difficult to customize.
- Require training of maintenance staff
- May be difficult to use with existing system

8.8 Characteristics of a successful CASE Tool:

A CASE tool must have the following characteristics in order to be used efficiently:

A standard methodology: A CASE tool must support a standard software development methodology and standard modeling techniques. In the present scenario most of the CASE tools are moving towards UML.

Flexibility: Flexibility in use of editors and other tools. The CASE tool must offer flexibility and the choice for the user of editors' development environments.

Strong Integration: The CASE tools should be integrated to support all the stages. This implies that if a change is made at any stage, for example, in the model, it should get reflected in the code

documentation and all related design and other documents, thus providing a cohesive environment for software development.

Integration with testing software: The CASE tools must provide interfaces for automatic testing tools that take care of regression and other kinds of testing software under the changing requirements.

Support for reverse engineering: A CASE tools must be able to generate complex models from already generated code.

8.9 Reasons for using CASE tools:

The primary reasons for employing a CASE tool are:

- to extend productivity
- to assist turn out higher quality code at a lower price

8.10 SELF ASSESSMENT TEST

1. What are the CASE tools?
2. What are various types of CASE tools? Explain briefly.
3. Explain architecture of CASE tool.
4. What are components of CASE tools? Explain briefly.
5. Write a note on CASE environment. Hoe it is differ from programming environment?
6. What are the characteristics of good CASE tool?

SUBJECT: Software Engineering	
COURSE CODE: MCA 14	AUTHOR: Prof Pradeep Kumar Bhatia
LESSON NO. 9	

Software Quality

Structure

9.0 Learning Objective

9.1 Software Quality Concepts

9.1.1 Software Quality

9.1.2 Software Quality Assurance

9.1.3 Software Quality Control

9.2 ISO 9126 Quality Factor

9.3 McCall's Quality Model

9.4 Software Review

9.4.1 Walkthroughs

9.4.2 Technical Review

9.4.3 Inspection

9.4.4. Roles and Responsibilities in a Review

9.4.5 Phases of a formal Review

9.5 Defect Amplification Model

9.6 ISO 9000 series Quality Standards

9.7 Capability Maturity Model (CMM)

9.8 Software Reliability.

9.0 Learning Objective

Tis lesson focus on quality measurement, quality control and to quality improvement with various SQA activities Walkthrough, Audit, Inspection. Quality stands ISO and CMM are covered in this lesson . This lesson also help provide reliable software and find out the relationship of reliability with testing and quality..

9.1 Software Quality Concepts

9.1.1 Software Quality

Quality is extremely hard to define. Still, there are several definitions for quality. Different authors define the term differently. We give below some definitions of quality.

Definitions of Quality

I Quality = Zero defects (Philip B. Crosby)

II Quality = "conformance to requirements" (Philip B. Crosby-1979)

III Quality = "fitness for use" (Juran and Gryna -1970)

Conformance to requirements

- It implies that requirements must be clearly stated that they cannot be misunderstood. Then in the development and production process , measurements are taken regularly to determine conformance to these requirements.
- The nonconformance are regarded as *defects* "*the absence of quality"*

Fitness for use

- It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.
- Extent to which the product conforms to the intent of the design.

Fitness for use implies more significant role for customers' requirements.

IV ISO(Indian Organization for Standardization) 1986

" The totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs"

V IEEE 610.12-1998:

"A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements"

VI "Quality is whatever the customer or client says it"

The above definitions cannot be applied to software. However some definitions have been suggested which specifically refer to software.

Definitions of Software quality

Definition I

Software quality as '**fitness for needs'**. Kitchenham(1989)

This definition involves two features of a piece of quality software :

- Conformance to its specification
- Fitness for its intended purpose

"Quality of a system is Definition II

equivalent to defining and measurement of list of software quality attributes required for the system"

Definition III

"Software quality is the degree to which a system, component or process meets specified requirements"

Definition IV

IEEE Standard (IEEE Std 729-1983)

"The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer"

V "Quality is whatever the customer or client says it"

Why software quality different from other types of quality?

Gillies A.C. 1990 in a workshop held at Institutes of Salfrid, in 1990**.**

- Each type of product made its own demands.
- Computer software is problematical in nature for the following reasons:
  - Software has no physical existence
  - The lack of knowledge of client needs at the start
  - The change client needs over the time
  - The rapid change in hardware and software
  - The high expectations of customers.

9.1.2 Quality Assurance

First we will understand the term Assurance

Assurance is nothing but a positive declaration on a product or service, which gives confidence. It is certainty of a product or a service, which it will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

Quality Assurance

Definition I

Quality Assurance in Software Testing is defined as a procedure to ensure the quality of software products or services provided to the customers by an organization. Quality assurance focuses on improving the software development process and making it efficient and effective as per the quality standards defined for software products. Quality Assurance is popularly known as QA

Definition II

It is defined as an activity to ensure that an organization is providing the best possible product or service to customers.

How to do Quality Assurance: Complete Process?

Quality Assurance methodology has a defined cycle called PDCA cycle or Deming cycle. The phases of this cycle are:

- **Plan**
- **Do**
- **Check**
- **Act**

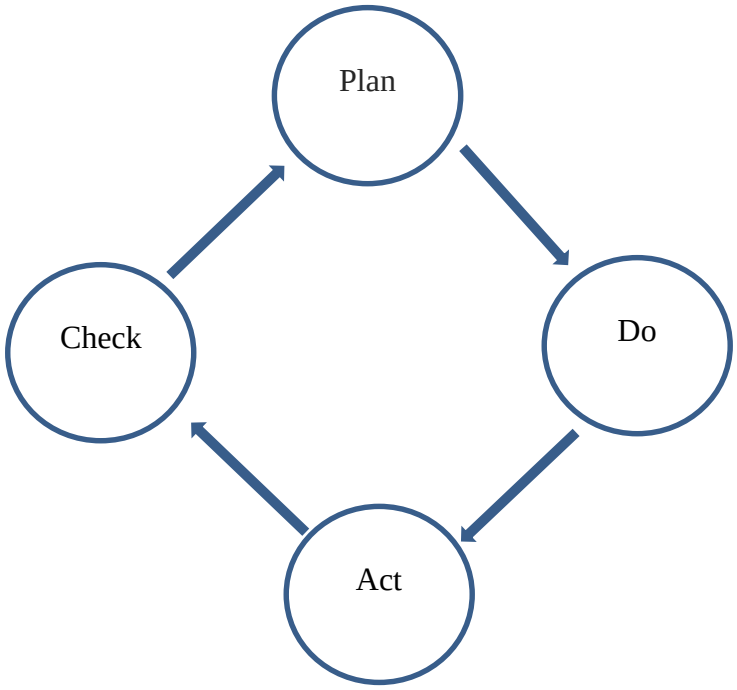


Figure 9.1 PDCA Plan

These above steps are repeated to ensure that processes followed in the organization are evaluated and improved on a periodic basis. Let's look into the above QA Process steps in detail -

- **Plan -** Organization should plan and establish the process related objectives and determine the processes that are required to deliver a high-Quality end product.
- **Do** - Development and testing of Processes and also "do" changes in the processes
- **Check -** Monitoring of processes, modify the processes, and check whether it meets the predetermined objectives

- **Act -** A Quality Assurance tester should implement actions that are necessary to achieve improvements in the processes

An organization must use Quality Assurance to ensure that the product is designed and implemented with correct procedures. This helps reduce problems and errors, in the final product.

9.1.2 Quality Control

Definition I

Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.

Definition II

Quality Control in Software Testing is a systematic set of processes used to ensure the quality of software products or services. The main purpose of the quality control process is ensuring that the software product meets the actual requirements by testing and reviewing its functional and non-functional requirements. Quality control is popularly abbreviated as QC

The main aim of Quality control is to check whether the products meet the specifications and requirements of the customer. If an issue or problem is identified, it needs to be fixed before delivery to the customer.

QC also evaluates people on their quality level skill sets and imparts training and certifications. This evaluation is required for the service based organization and helps provide "perfect" service to the customers.

Difference between Quality Control and Quality Assurance

Sometimes, QC is confused with the QA. Quality control is to examine the ***product or service*** and check for the result. Quality Assurance in Software Engineering is to examine the *processes* and make changes to the processes which led to the end-product. As shown in figure 9.2 .

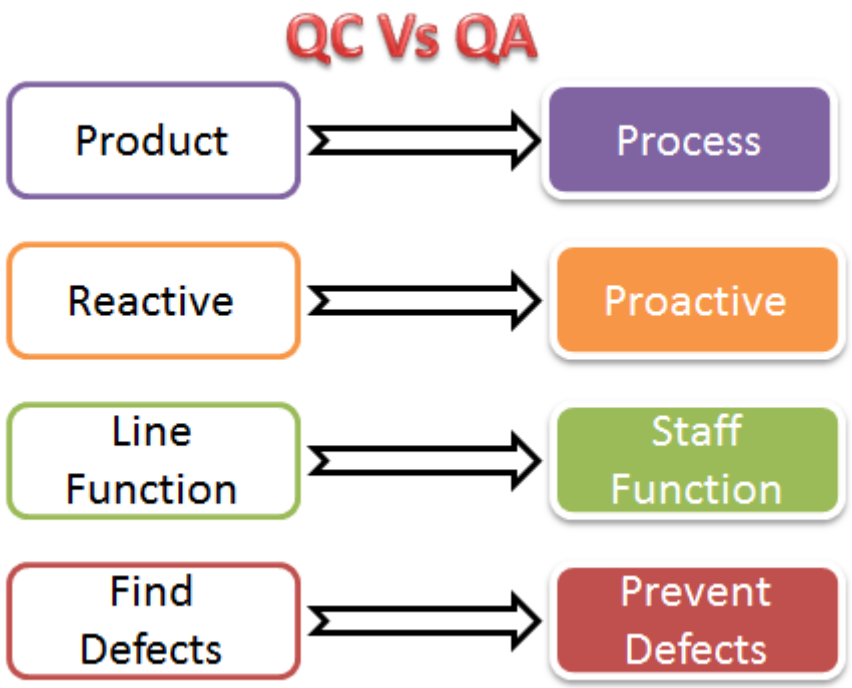


Figure 9.2 Difference between QC and QA

Difference between Quality Assurance (QA) and Quality Control (QC)

Quality Assurance (QA)	Quality Control (QC)
• It is a procedure that focuses on providing assurance that quality requested will be achieved	• It is a procedure that focuses on fulfilling the quality requested.
• QA aims to prevent the defect	• QC aims to identify and fix defects
• It is a method to manage the quality- Verification	• It is a method to verify the quality- Validation
• It does not involve executing the program	• It always involves executing a program
• It's a Preventive technique	• It's a Corrective technique

• It's a Proactive measure	• It's a Reactive measure
• It is the procedure to create the deliverables	• It is the procedure to verify that deliverables
• QA involves in full software development life cycle	• QC involves in full software testing life cycle
• In order to meet the customer requirements, QA defines standards and methodologies	• QC confirms that the standards are followed while working on the product
• It is performed before Quality Control	• It is performed only after QA activity is done
• It is a Low-Level Activity, it can identify an error and mistakes which QC cannot	• It is a High-Level Activity, it can identify an error that QA cannot
• Its main motive is to prevent defects in the system. It is a less time-consuming activity	• Its main motive is to identify defects or bugs in the system. It is a more time-consuming activity
• QA ensures that everything is executed in the right way, and that is why it falls under verification activity	• QC ensures that whatever we have done is as per the requirement, and that is why it falls under validation activity
• It requires the involvement of the whole team	• It requires the involvement of the Testing team
• The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)	• The statistical technique applied to QC is known as SQC or Statistical Quality Control

QC and QA activities

Examples of QC and QA activities are as follows:

Quality Control Activities	Quality Assurance Activities
Walkthrough	Quality Audit
Testing	Defining Process
Inspection	Tool Identification and selection
Checkpoint review	Training of Quality Standards and Processes

The above activities are concerned with Quality Assurance and Control mechanisms for any product and not essentially software. With respect to software

- QA becomes SQA (Software Quality Assurance)
- QC becomes Software Testing.

Differences between SQA and Software Testing

Following table explains on differences between SQA and Software Testing:

SQA	Software Testing
Software Quality Assurance is about engineering process that ensures quality Involves activities related to the implementation of processes, procedures, and standards. Example - Audits Training	Software Testing is to test a product for problems before the product goes live Involves actives concerning verification of product Example - Review Testing

Process focused	Product focused
Preventive technique	Corrective technique
Proactive measure	Reactive measure
The scope of SQA applied to all products that will be created by the organization	The scope of Software Testing applies to a particular product being tested.

Most people get confused when it comes to pin down the differences among Quality Assurance, Quality Control, and Testing. Although they are interrelated and to some extent, they can be considered as same activities, but there exist distinguishing points that set them apart. The following table lists the points that differentiate QA, QC, and Testing.

Quality Assurance	Quality Control	Testing
QA includes activities that ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	It includes activities that ensure the verification of a developed software with respect to documented (or not in some cases) requirements.	It includes activities that ensure the identification of bugs/error/defects in a software.
Focuses on processes and procedures rather than conducting actual testing on the system.	Focuses on actual testing by executing the software with an aim to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
Process-oriented activities.	Product-oriented activities.	Product-oriented activities.

Preventive activities.	It is a corrective process.	It is a preventive process.
It is a subset of Software Test Life Cycle (STLC).	QC can be considered as the subset of Quality Assurance.	Testing is the subset of Quality Control.

9.2 ISO/IEC 9126 Quality Factors

This standard deals with the following aspects to determine the quality of a software application −

- Quality model
- External metrics
- Internal metrics
- Quality in use metrics

This standard presents some set of quality attributes for any software such as −

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

The above-mentioned quality attributes are further divided into sub-factors.

The quality model presented in the first part of the standard, ISO/IEC 9126-1, classifies software quality in a structured set of characteristics and sub-characteristics as follows:

- **Functionality** - "A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs."
    - Suitability
    - Accuracy

  - Interoperability
  - Security
  - Functionality compliance
- Reality - "A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time."
  - Maturity
  - Fault tolerance
  - Recoverability
  - Reliability compliance
- Usability - "A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users."
  - Understandability
  - Learnability
  - Operability
  - Attractiveness
  - Usability compliance
- Efficiency - "A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions."
  - Time behaviour
  - Resource utilization
  - Efficiency compliance
- Maintainability - "A set of attributes that bear on the effort needed to make specified modifications."
  - Analyzability

  - Changeability
  - Stability
  - Testability
  - Maintainability compliance
- Portability - "A set of attributes that bear on the ability of software to be transferred from one environment to another."
  - Adaptability
  - Installability
  - Co-existence
  - Replaceability
  - Portability compliance

Each quality sub-characteristic (e.g. adaptability) is further divided into attributes. An attribute is an entity which can be verified or measured in the software product. Attributes are not defined in the standard, as they vary between different software products.

Software product is defined in a broad sense: it encompasses executable, source code, architecture descriptions, and so on. As a result, the notion of user extends to operators as well as to programmers, which are users of components such as software libraries.

The standard provides a framework for organizations to define a quality model for a software product. On doing so, however, it leaves up to each organization the task of specifying precisely its own model. This may be done, for example, by specifying target values for quality metrics which evaluates the degree of presence of quality attributes.

Internal Metrics

Internal metrics are those which do not rely on software execution (static measure).

External metrics

External metrics are applicable to running software.

Quality-in-use metrics

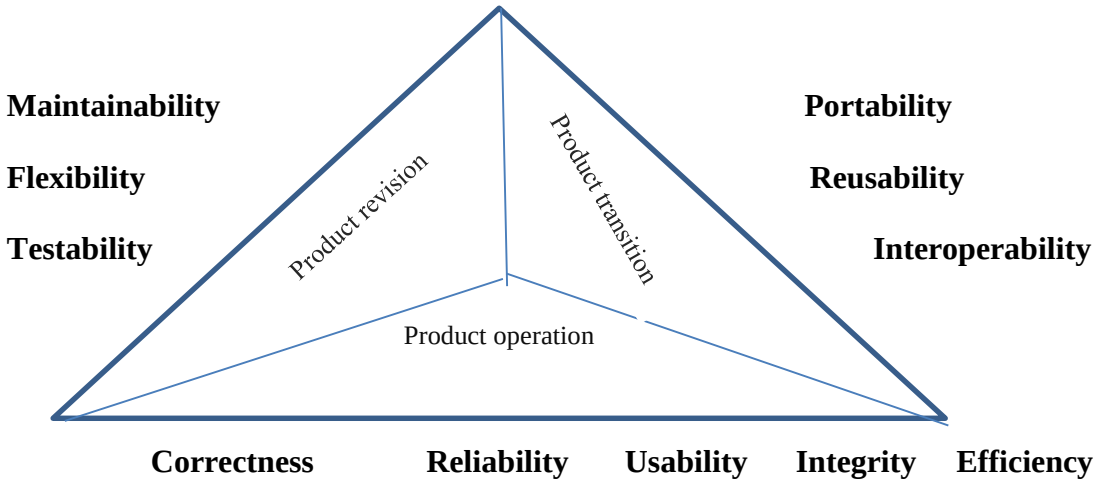
Quality-in-use metrics are only available when the final product is used in real conditions. Ideally, the internal quality determines the external quality and external quality determines quality in use.

ISO/IEC 9126 distinguishes between a defect and a nonconformity, a **defect** being "The nonfulfillment of intended usage requirements", whereas a **nonconformity** is "The nonfulfillment of specified requirements". A similar distinction is made between validation and verification, known as V&V in the testing trade.

.9.3 McCall's Quality Model

McCall and Cavano in 1977 defined number of factors that measures software quality. These factors assess software from three distinct point of view:

- **Product operation(using it)**
- **Product revision(changing it)**
- **Product transition (modifying it to work in a different environment ie porting it)**



9.3 Architecture of McCall's Model

Referring to the factors noted in fig 9.3 McCall provides the following descriptions:

Product Operation

Correctness: The extent to which a program satisfies its specifications and fulfil the customer's mission objectives.

Reliability: It is the property which defines how well he software meets its requirements.

Efficiency: The amount of computing resources and code required by a program to perform its function.

Usability : The effort required to learn , operate , prepare input , and interpret output of a program.

Integrity: The extent to which access the software or data unauthorised persons. can be controlled .

Product Revision

Maintainability: It is the effort required to locate and fix errors in operating programs.

Flexibility: The effort to modify an operational program.

Testability: The effort required to test a program to ensure that it performs its intended function.

Product Transition

Portability: The effort required to transfer the program from one hardware and/or software system environment to another.

Reusability: The extent to which parts of the software reused in other related applications.

Interoperability: The effort required to couple one system to another.

9.4 Review

A review is a systematic examination of a document by one or more people with the main aim of finding and removing errors early in the software development life cycle. Reviews are used to verify documents such as requirements, system designs, code, test plans and test cases. These are powerful way to improve the quality and productivity of software development to recognize and fix their own defects early in the software development process.

Types of Review

9.4.1 Walkthrough

9.4.2 Technical Review

9.4.3 Inspection

9.4.1 Walkthrough

It is informal review

It is managed by authors.It is especially useful for higher-level documents, such as requirement specifications and architectural documents.

This is especially useful for people if they are not from the software discipline , who are not used to, or cannot easily understand software.

Author and participants through the document according to his or her thought processes, to achieve a common understanding and to gather feedback.

9.4.2 Technical Review

- It is less formal review
- it is led by a trained moderator, but also can be led by a technical expert.
- It is often performed as a peer review without management participation.
- Defects are found by the experts (such as architects , designers, key users) who focus on the content of the document.
- It vary from quit informal review to very informal review.

9.4.3 Inspection

- It is the most formal review type.
- It is usually led by a trained moderator (certainly not by the author).
- The document under inspection is prepared and checked thoroughly by the reviewers before the meeting,
- It involves peers to examine the product
- A separate preparation is carried out during which product is examined an defects are found and defects found are documented in logging list

- A formal follow-up is carried out by moderator apply exit criteria. inspections can be balanced to serve a number of goals.

9.4.4. Roles and Responsibilities in a Review

a) The moderator : The moderator (or review leader) leads the review process. His role is to determine the type of review, approach and the composition of the review team.

The moderator also schedules the meeting, spreads documents before the meeting, trainers other team members, leads possible discussions and stores the data that is collected.

b) The author : As the writer of the 'document under review', the author's basic goal should be to learn as much as possible with regard to improving the quality of the document.

The author's task is to illuminate unclear areas and to understand the defects found.

c) The scribe/ recorder : The scribe (or recorder) has to record each defect found and any suggestions or feedback given in the meeting for process improvement.

d) The reviewer: The role of the reviewers is to check defects and further improvements in accordance to the business specifications, standards and domain knowledge.

e) The manager : Manager is involved in the reviews as he or she decides on the execution of reviews, allocates time in project schedules and determines whether review process objectives have been met or not.

9.4.5 Phases of a formal Review

Formal review is well structured and regulated . A formal review process consists of six main steps: Planning, Overview Preparation, Inspection Meeting, Rework, and Follow-up.

a) **Planning** The review process for a particular review begins with a 'request for review' by the author to the moderator (or inspection leader). A moderator is often assigned to take care of the scheduling (dates, time, place and invitation) of the review. The project planning needs to allow time for review and rework activities, thus providing engineers with time to thoroughly participate in reviews. There is an entry check performed on the documents and it is decided that which documents are to be considered or not. The

document size, pages to be checked, composition of review team, roles of each participant, strategic approach are decided into planning phase.

b) Kick-Off The goal of this meeting is to get everybody on the same page regarding the document under review. Also the result of the entry and exit criteria are discussed. Basically, During the kick-off meeting, the reviewers receive a short introduction on the objectives of the review and the documents. Role assignments, checking rate, the pages to be checked, process changes and possible other questions are also discussed during this meeting. Also, the distribution of the document under review, source documents and other related documentation, can also be done during the kick-off.

c) Preparation In this phase, participants work individually on the document under review using the related documents, procedures, rules and checklists provided. The individual participants identify defects, questions and comments, according to their understanding of the document and role. Spelling mistakes are recorded on the document under review but not mentioned during the meeting. The annotated document will be given to the author at the end of the logging meeting. Using checklists during this phase can make reviews more effective and efficient.

d) Review Meeting This meeting typically consists of the following elements:-
logging phase During the logging phase the issues, e.g. defects, that have been identified during the preparation are mentioned page by page, reviewer by reviewer and are logged either by the author or by a scribe.

- **discussion phase** The issues classified as discussion items will be handled during discussion phase. Participants can take part in the discussion by bringing forward their comments and reasoning.
- **decision phase**. At the end of the meeting, a decision on the document under review has to be made by the participants, sometimes based on formal exit criteria. The most important exit criterion is the average number of critical and major defects found per page. If the number of defects found per page exceeds a certain level, the document must be reviewed again, after it has been reworked. If the document complies with the exit criteria, the document will be checked during follow-up by the moderator or one or more participants. Subsequently, the document can leave or exit the review process.

e) Rework Based on the defects detected and improvements suggested in the review meeting, the author improves the document under review. In this phase the author would be doing all the rework to ensure that defects detected should fixed and corrections should be properly implied. Changes that are made to the document should be easy to identify during follow-up, therefore the author has to indicate where changes are made.

f) Follow-Up After the rework, the moderator should ensure that satisfactory actions have been taken on all logged defects, improvement suggestions and change requests. If it is decided that all participants will check the updated document, the moderator takes care of the distribution and collects the feedback. In order to control and optimize the review process, a number of measurements are collected by the moderator at each step of the process. Examples of such measurements include number of defects found; number of defects found per page, time spent checking per page, total review effort, etc. It is the responsibility of the moderator to ensure that the information is correct and stored for future analysis.

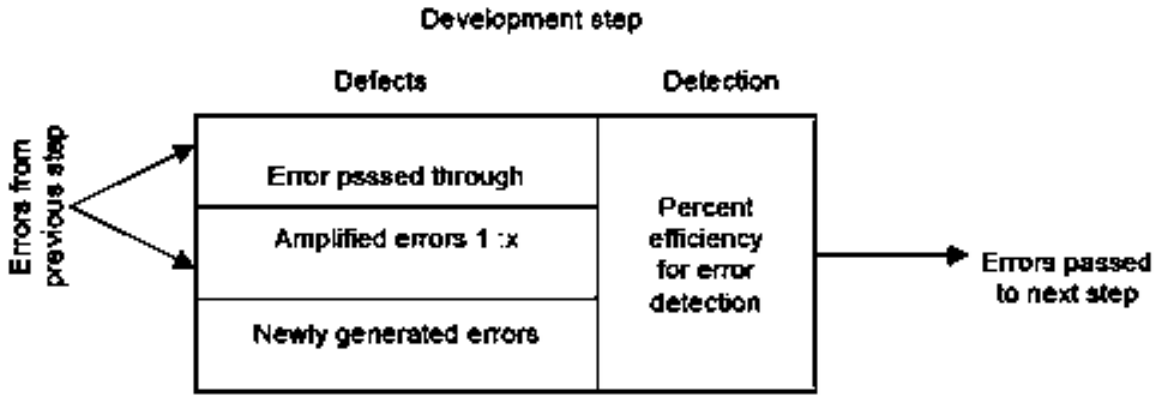
9.5 Defect Amplification and Removal

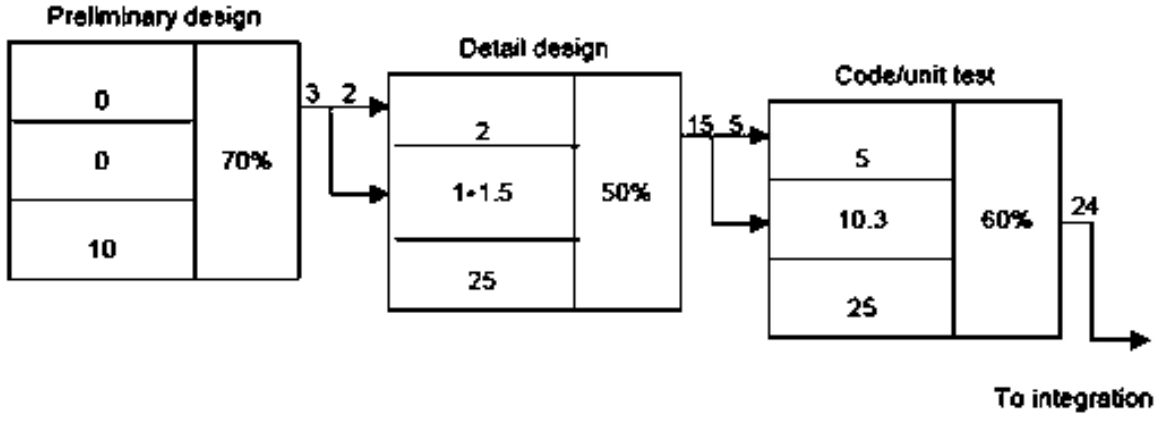
The defect amplification model can be used to describe the detection and generation of errors during preliminary design, detail design and coding steps of the software engineering procedure. The model is describing schematically in the given Figure 9.2 a box is represents a software development steps. In the duration of these steps the step, errors may be inadvertently occurring. Review may fail to uncover newly generated errors and errors from previous steps resulting in some number of errors which are passed by. Some cases errors passed by from earlier steps which are amplified (amplification factor, x) through current work. The box subdivisions represent each of these characteristics and the percent efficiency for detecting bugs a function of the thoroughness of review.

In the Figure 9.4 describes a hypothetical example of defect amplification for a software development procedure in that no reviews are conducted. Describe in the figure each test step is supposed to uncover and correct 50 % of all incoming errors without introducing any new errors an optimistic assumption 10 preliminary design errors are amplified to 94 errors previous testing commences. 12 latent defects are released to the area. The Figure 9.4 considers the similar conditions except which code and design reviews are conducted as categorised of each development step. In this case ten initial preliminary design errors are amplified to 24 errors previously testing commences. Only 3 latent defects exist.

Through recalling the relative costs related with the correction and discovery of errors overall cost with and without review for your hypothetical example can be build. In the Table it can be seen which total cost for maintenance and development when reviews are organized is 783 cost units when no reviews are conducted total cost is 2177 units-nearly 3 times more costly.

To conduct reviews a developer must expend effort and time and the development company must spend money. Moreover, the results of the preceding example leave little doubt which we have encountered a pay now or pay much more lately syndrome. The formal technical reviews for design and other technical activities give a demonstrable cost advantages. They should be organized





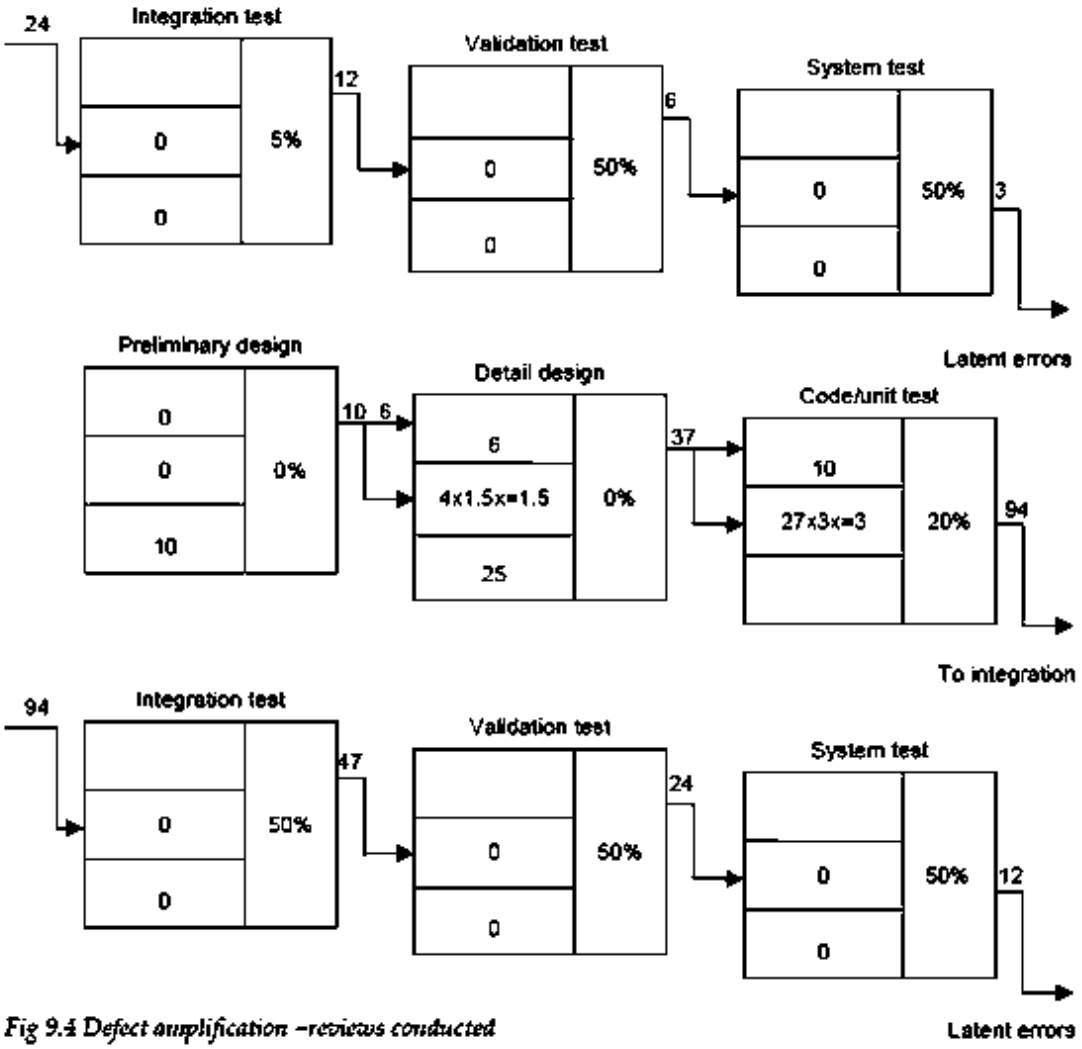


Fig 9.4 Defect amplification –reviews conducted

9.6 ISO 9000 Series Quality Standards

ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. This standard was first established in 1987, and it is related to Quality Management Systems. This helps the organization ensure quality to their customers and other stakeholders. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

a)Types of ISO 9000 Quality Standards

ISO 9000 is a series of three standards :

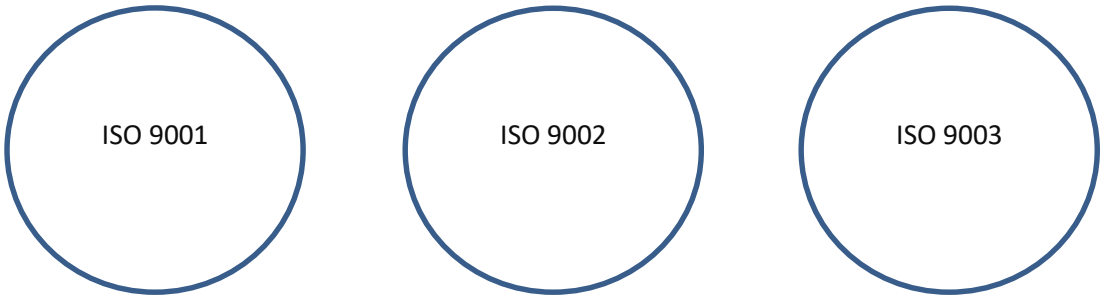


Figure 9.5 ISO 9000 Series of Quality Standards

The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.

1. ISO 9001: This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.
2. ISO 9002: This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.
3. ISO 9003: This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.

b) Process to get ISO 9000 certification

An organization who wishes to be certified as ISO 9000 is audited based on their functions, products, services and their processes. The main objective is to review and verify whether the organization is

following the process as expected and check whether existing processes need improvement. This certification helps -

- Increase the profit of the organization
- Improves Domestic and International trade
- Reduces waste and increase the productivity of the employees
- Provide Excellent customer satisfaction

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:

Stage 1: Application: Once an organization decided to go for ISO certification, it applies to the registrar for registration.

Stage 2: Pre-Assessment: During this stage, the registrar makes a rough assessment of the organization.

Stage 3 : Document review and Adequacy of Audit: During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.

Stage 4: Compliance Audit: During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.

Stage 5: Registration: The Registrar awards the ISO certification after the successful completion of all the phases.

Stage 6: Continued Inspection: The registrar continued to monitor the organization time by time.

Many organizations around the globe develop and implement different standards to improve the quality needs of their software. This chapter briefly describes some of the widely used standards related to Quality Assurance and Testing.

9.7 CMM

CMM was developed and is promoted by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987, a research and development centre promote by the U.S. Department of Defense (DOD).

The term "maturity" relates to the degree of formality and optimization of processes, from ad-hoc practices, to formally defined steps, to managed result metrics, to active optimization of the processes.

The model's aim is to improve existing software development processes, but it can also be applied to other processes.

The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes. Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

Key Process Areas (KPA's):

Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.

Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.

The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

a) Levels of CMM

CMM has 5 levels. An organization is certified at CMM level 1 to 5 based on the maturity of their Quality Assurance Mechanisms. Figure 9.6 shows architecture of CMM.

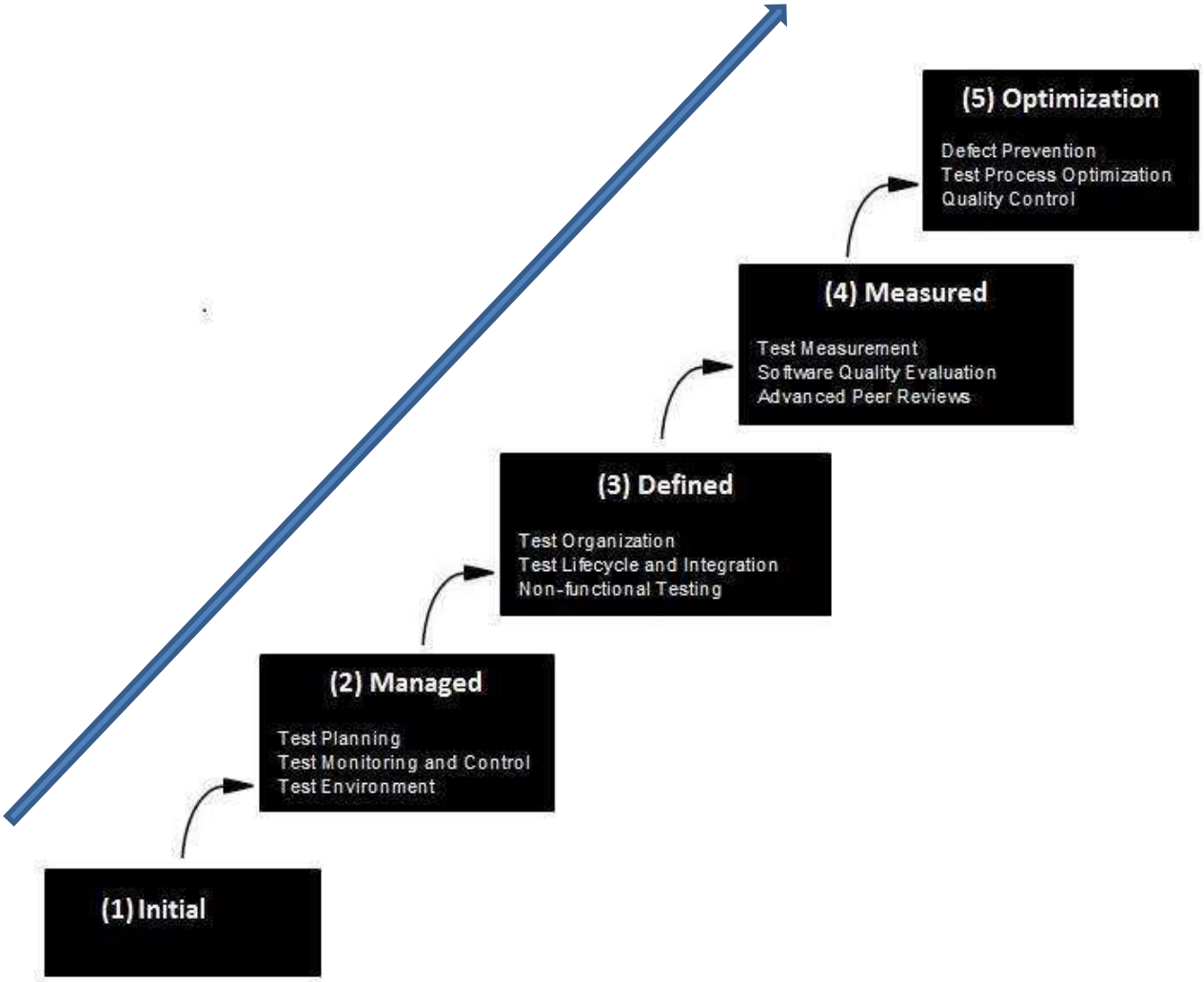


Figure 9.6 Architecture of CMM

- Level 1 - Initial: In this stage the quality environment is unstable. Simply, no processes have been followed or documented
- Level 2 - Repeatable: Some processes are followed which are repeatable. This level ensures processes are followed at the project level.
- Level 3 - Defined: Set of processes are defined and documented at the organizational level. Those defined processes are subject to some degree of improvement.
- Level 4 - Managed: This level uses process metrics and effectively controls the processes that are followed.

- Level 5 - Optimizing: This level focuses on the continuous improvements of the processes through learning & innovation.

Table 9.1 depicts CMM Level with various KPA's.

Table 9.1

CMM Level	Focus	Key Process Areas
1. Initial	Competent People	NO KPA'S
2. Repeatable	Project Management	Software Project Planning software Configuration Management
3. Defined	Definition of Processes	Process definition Training Program Peer reviews
4. Managed	Product and Process quality	Quantitiative Process Metrics Software Quality Management
5. Optimizing	Continuous Process improvement	Defect Prevention Process change management Technology change management

The focus of each SEI CMM level and the Corresponding Key process areas.

In 2006, the Software Engineering Institute at Categie Mellon University developed the Capability Maturity Model Integration, which has largely superseded the CMM .

The Capability Maturity Model Integrated (CMMI) is a process improvement approach developed specially for software process improvement. It is based on the process maturity framework and used as a general aid in business processes in the Software Industry. This model is highly regarded and widely used in Software Development Organizations.

c) Test Maturity Model (TMM):

This model assesses the maturity of processes in a Testing Environment. Even this model has 5 levels, defined below-

- Level 1 - Initial: There is no quality standard followed for testing processes and only ad-hoc methods are used at this level
- Level 2 - Definition: Defined process. Preparation of test strategy, plans, test cases are done.
- Level 3 - Integration: Testing is carried out throughout the software development lifecycle (SDLC) - which is nothing but integration with the development activities, E.g., V- Model.
- Level 4 - Management and Measurement: Review of requirements and designs takes place at this level and criteria has been set for each level of testing
- Level 5 - Optimization: Many preventive techniques are used for testing processes, and tool support(Automation) is used to improve the testing standards and processes.

9.8 SOFTWARE RELIABILITY

Definitions

Software reliability is the probability of the software causing a system failure over some specified operating time.

Software does not fail due to wear out but does fail due to faulty functionality, timing, sequencing, data, and exception handling. The software fails as a function of operating time as opposed to calendar time.

Software reliability is defined as the probability of failure-free operation of a software system for a specified time in a specified environment.

The key elements of the definition include probability of failure-free operation, length of time of failure-free operation and the given execution environment. Failure intensity is a measure of the reliability of a software system operating in a given environment

• A Formal Definition***: Reliability is the probability of failure-free operation of a system over a specified time within a specified environment for a specified purpose.***

An Informal definition: ***Reliability is a measure of how closely a system matches its stated specification.***

• Another Informal Definition: ***Reliability is a measure of how well the users perceive a system provides the required services.***

Software Reliability is an essential connect of software quality,

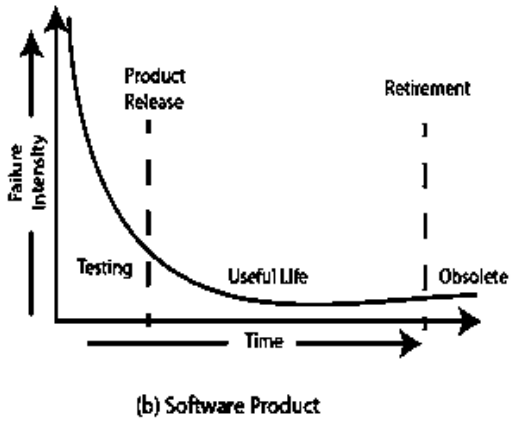
a) Hardware vs. Software Reliability

Hardware Reliability	Software Reliability
Hardware faults are mostly physical faults.	Software faults are design faults, which are tough to visualize, classify, detect, and correct.
Hardware components generally fail due to wear and tear.	Software component fails due to bugs.
In hardware, design faults may also exist, but physical faults generally dominate.	In software, we can simply find a strict corresponding counterpart for "manufacturing" as the hardware manufacturing process, if the simple action of uploading software modules into place does not count. Therefore, the quality of the software will not change once it is uploaded into the storage and start running
Hardware exhibits the failure features shown in the following figure:9.1(a) It is called the bathtub curve. Period A, B, and C stand for burn-in phase, useful life phase, and end-of-life phase respectively.	Software reliability does not show the same features similar as hardware. A possible curve is shown in the following figure: 9.1(b) If we projected software reliability on the same axes.

There are two significant differences between hardware and software curves are:

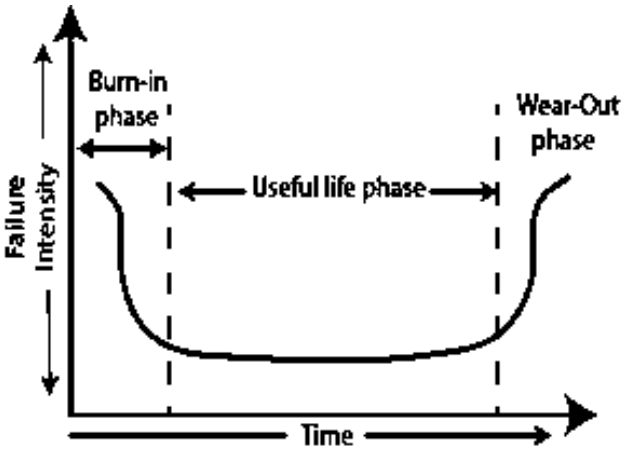
One difference is that in the last stage, the software does not have an increasing failure rate as hardware does. In this phase, the software is approaching obsolescence; there are no motivations for any upgrades or changes to the software. Therefore, the failure rate will not change.

The second difference is that in the useful-life phase, the software will experience a radical increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects create and fixed after the updates.



(b) Software Product

Figure 9.7(a)



(a) Hardware Product

Figure 9.7(b)

b) Reliability metrics

1) Mean Time to Failure (MTTF)

MTTF is described as the time interval between the two successive failures. An MTTF of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. MTTF is consistent for systems with large transactions.

For example, It is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.

To measure MTTF, we can evidence the failure data for n failures. Let the failures appear at the time instants t_1,t_2.....t_n.

MTTF can be calculated as

$$\sum_{i=1}^{n} \frac{t_{i+1}-t_i}{(n-1)}$$

2) . Mean Time to Repair (MTTR)

Once failure occurs, some-time is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean Time Between Failure (MTBR)

We can merge MTTF & MTTR metrics to get the MTBF metric.

MTBF = MTTF + MTTR

Thus, an MTBF of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in MTTF.

4. Rate of occurrence of failure (ROCOF)

It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. ROCOF is the frequency of occurrence with which unexpected role is likely to appear. A ROCOF of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric.

5. Probability of Failure on Demand (POFOD)

POFOD is described as the probability that the system will fail when a service is requested. It is the number of system deficiency given several systems inputs.

POFOD is the possibility that the system will fail when a service request is made.

A POFOD of 0.1 means that one out of ten service requests may fail. POFOD is an essential measure for safety-critical systems. POFOD is relevant for protection systems where services are demanded occasionally.

6. Availability (AVAIL)

Availability is the probability that the system is applicable for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is feasible to be available for 995 of these. The percentage of time that a system is applicable for use, taking into

account planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its AVAIL is 96%.

Self Assessment Test

1. Define the terms Software quality., Quality control, Software Quality, Assurance. How these are related to each other?
2. Describe McCall's Quality Model.
3. Write short note on
    a) Review
    b) Inspection
    c) Walk through
4. What do you meant by quality standards? Describe CMM and ISO . Compare CMM and ISO.
5. Define the term Software Reliability. What are the various reliability metrics? Explain briefly.

****

NOTES

NOTES

NOTES

NOTES

NOTES