Master of Computer Application

(Open and Distance Learning Mode)

Semester – II

Software Engineering

Centre for Distance and Online Education (CDOE)

DEVI AHILYA VISHWAVIDYALAYA, INDORE

"A+" Grade Accredited by NAAC

IET Campus, Khandwa Road, Indore - 452001

www.cdoedavv.ac.in

www.dde.dauniv.ac.in

Program Coordinator

Dr. Anand More
School of Computer Science and IT
Devi Ahilya Vishwavidyalaya, Indore – 452001

Content Design Committee

Dr. Pratosh Bansal
Centre for Distance and Online Education
Devi Ahilya Vishwavidyalaya, Indore – 452001

Dr. C.P. Patidar
Institute of Engineering & Technology
Devi Ahilya Vishwavidyalaya, Indore – 452001

Dr. Shaligram Prajapat
International Institute of Professional Studies
Devi Ahilya Vishwavidyalaya, Indore – 452001

Language Editors

Dr. Arti Sharan
Institute of Engineering & Technology
Devi Ahilya Vishwavidyalaya, Indore – 452001

Dr. Ruchi Singh
Institute of Engineering & Technology
Devi Ahilya Vishwavidyalaya, Indore – 452001

SLM Author(s)

Mrs. Sunita Goud
M. Tech.
SCS, Devi Ahilya Vishwavidyalaya, Indore – 452001

Mr. Vikas Vankhede
B.E., M.E.
IET, Devi Ahilya Vishwavidyalaya, Indore – 452001

Copyright : Centre for Distance and Online Education (CDOE), Devi Ahilya Vishwavidyalaya
Edition : 2022 (Restricted Circulation)
Published by : Centre for Distance and Online Education (CDOE), Devi Ahilya Vishwavidyalaya
Printed at : University Press, Devi Ahilya Vishwavidyalaya, Indore – 452001

SOFTWARE ENGINEERING

Table of Content

Lesson-3: SOFTWARE PROJECT MANAGEMENT

3.0 Objectives

3.1 Introduction

3.2 Software Project

3.3 Software Project Management

- 3.3.1 Software Project Management Activities.
- 3.3.2 Software Project Estimation Parameters

3.4 Project Scheduling Techniques

- 3.4.1 GANTT CHART
- 3.4.2 PERT CHART
- 3.4.3 CPM CHART
- 3.4.4 Work Breakdown Structure

3.5 SPMP

3.6 Summary

3.7 Suggestions

3.8 Model Questions

Lesson-4: SOFTWARE PROJECT ESTIMATION AND RISK MANAGEMENT

4.0 Objectives

4.1 Introduction

4.2 Decomposition Techniques

- 4.2.1 Problem Based Estimation
- 4.2.2 Process Based Estimation

4.3 Software Risk Management

- 4.3.1 Risk Management Activities
- 4.3.2 Risk Identification
- 4.3.3 Risk Analysis
- 4.3.4 Risk classification

4.4 Strategies for Risk Management

4.5 Summary

4.6 Suggested Readings

4.7 Model Questions

Lesson-5: SOFTWARE DESIGN

5.0 Objectives

5.1 Importance of Software Design Process.

5.2 Principles of Software Design Process.

5.3 Stages of design process

5.4 Design Failures

5.5 Design Remedies

- 5.5.1 Modularization
- 5.5.2 Concurrency
- 5.5.3 Coupling and Cohesion
- 5.5.4 Design Verification

5.6 Summary

5.7 Suggested Readings

5.8 Model Questions

Lesson-6: STRUCTURED ANALYSIS AND DESIGN TOOLS

6.0 Objectives

6.1 Introduction

6.2 Data Flow Diagram

6.2.1 Types of DFD

6.2.2 DFD Components

6.2.3 Levels of DFD

6.3 Data Dictionary

6.4 Entity Relationship Diagrams

6.5 Summary

6.6 Suggested Readings

6.7 Model Questions

Lesson -7: SOFTWARE TESTING

7.0 Introduction

7.1 Objectives of Software Testing

7.2 Principles of Software Testing

7.3 Testing Strategies

7.4 Testing Process

7.5 Black Box Testing

Lesson-8 SOFTWARE QUALITY AND MAINTENANCE

Lesson-1

SOFTWARE ENGINEERING FUNDAMENTALS

Structure

1.0 Objective

Software Engineering is the application of various engineering principles for design, development, testing, deployment and management of software.

1.1 Introduction

Software: Software is a logical entity rather than a physical system. It is a collection of codes and documents that does a specific job to satisfy a specific requirement.

Software is combination of

1. Instructions (computer programs) that when executed provide desired function and performance,
2. Data structures that enable the programs to adequately manipulate information, and
3. Documents that describe the operation and use of the programs.

Engineering is the development of products using best practices, principles and methods.

Definition: Software Engineering is the practical application of principles, skills and scientific knowledge for the design and construction of computer programs and documents required to develop, operate and maintain those programs.

It involves the process of engineering software and certification including requirements gathering, software design, construction and maintenance, configuration management, software development process, engineering models and methods, software quality, software engineering professional practices as well as foundational computing and mathematical engineering.

1.2 Levels of Software Engineering

There are three levels of software engineering:

1. **Operational software Engineering:** Software Engineering on the operational level focuses on how the software interacts with the system for its usability, functionality, dependability and security.
2. **Transitional Software Engineering:** It works on its scalability and flexibility. When the software is moved from one platform to another, the following factors deciding the software quality:

- Portability
- Interoperability
- Reusability
- Adaptability

3. **Recurrent Software Engineering:** It focuses on how the software functions with the existing system, as all parts of it to accommodate changes with the course of time.

1.3 Importance of Software Engineering

Software engineering is important because of few reasons:

1. **Rise of technology:** As technology is rising day by day, software engineering also has raised because of its usage in versatile areas of agriculture as well as medicine that made it incredibly important.
2. **Structural Formation:** The software engineering methodology has given everything in structured form from requirement analysis, design and coding to testing, So, the structured approach makes everything easier.
3. **Preventing Issues:** Software Engineering has helped to prevent issues such as quality assurance and user testing. So, it has become vitally important for the success of projects.
4. **Technology Enhancement:** Software Engineering play vital role to bring technologies forward. The entire Software as Service (SaaS) industry is aided with browser technology of software engineering.
5. **Automation and AI:** Artificial Intelligence/Machine Learning techniques have been widely used in software engineering to improve developer productivity, the quality of

software systems, and decision-making. For example, Computer Aided Design (CAD) and Computer Aided Machine (CAM) software projects are broadly used in engineering and design industry.

6. **Aiding Research:** New technologies arising in engineering industry. For example, new programming languages and new approaches to solve old problems has come from experienced developers.
7. **Perspective Future:** The future of many jobs and industries rests in the hands of software engineers. The Uber, software engineering project has shifted the entire transport industry. Software engineering is providing innovation in various areas that people have never imagined before.

1.4 Components of Software Engineering: A Layered Approach

Software engineering is a thorough investigation of engineering principles for the design, execution and maintenance of the software. So, to handle with the various issues of flexible, good quality and scalable software, its projects are designed through its various components as defined through layered technology.

Software engineering is totally a layered technology. Therefore, to develop software one will have to go from one layer to another layer. These layers are related to each other and fulfil all requirements of previous layer. The Figure1 basically shows that software engineering as a layered technology to develop software.

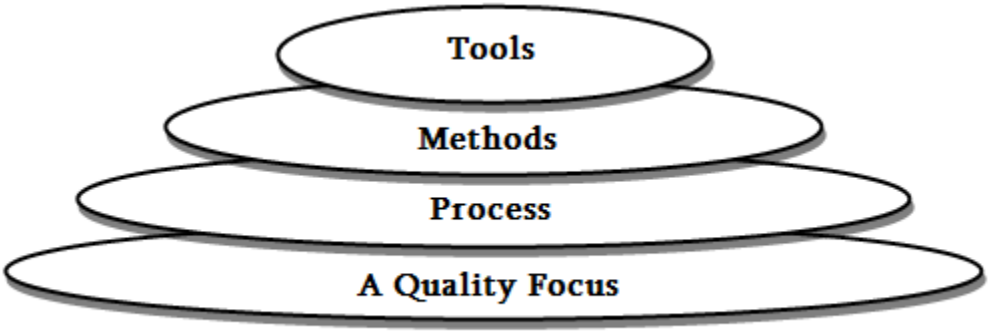


Figure: Flowchart of the Layers of Software Development

Figure1: Software Engineering as layered technology

Layered technology is divided into four parts:

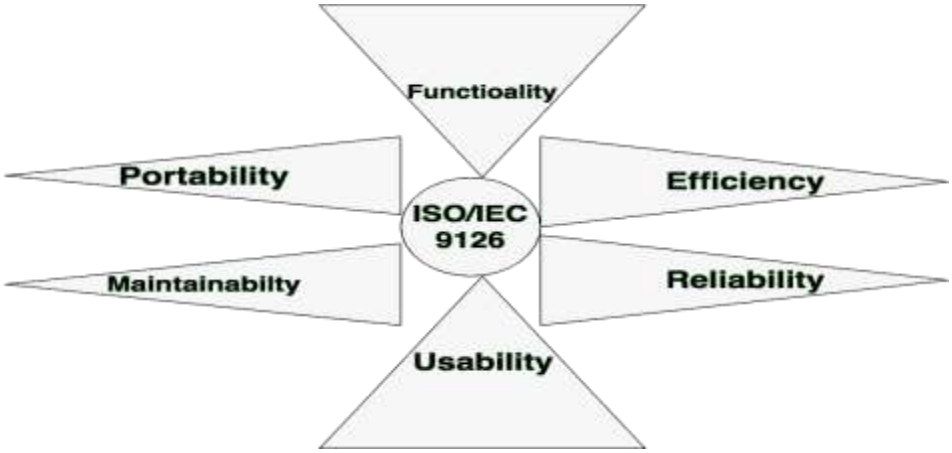
- **Tools:** Software engineering tools provide a self-operating system for processes and methods. Software engineering tools provide automated and semi-automated support for the process and its methods. The tools are integrated so that information created by one tool can be used by another. It serves the same purpose as a Computer Aided Design (CAD) or Computer Aided Engineering (CAE) for the computer hardware.
- **Methods:** Software engineering methods provide the technical details for building software. Methods encompass a broad array of tasks that include

requirements analysis, design, program construction, testing, and support. These methods rely on a set of basic principles to cover each area of the technology which include modelling activities and other descriptive techniques.

- **Process**: Software engineering process holds the technology to gather for the in-time delivery of the software. It defines a framework specifying various KPAs which are responsible for the management and control of software projects. They ensure that the appropriate technical methods and models are applied to generate the quality documents and data reports.
- **A Quality Focus**: Software engineering projects must be committed to quality**.** Hence various quality management principles are amended for the continuous process improvement for its future maintenance**.** It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

1.5 Software Characteristics:

Software is defined as a set of programs that enables the hardware to perform a specific task. All the programs that run by the computer are software. Software is a logical rather than a physical system element. Therefore, software has characteristics which are classified into six major components:



a. Functionality: It refers to the degree of performance of the software against its intended purpose.

Required functions to be verified are:

- Suitability
- Accuracy
- Interoperability
- Compliance
- Security

b. Efficiency: Software is said to be efficient if it uses the available resources in the most efficient manner. The software should perform whatever the user has demanded and give appropriate response in each case quickly.

c. Reliability: It is asset of attributes that specifies the capability of software to maintain its level of performance under the given condition for a stated period of time.

Required functions are:

- Recoverability
- Fault tolerance
- Maturity

d. Usability: It refers to the extent to which the software can be used with ease and the amount of effort or time required to learn how to use the software.

Required functions are:

- Understandability
- Learnability
- Operability

e. Maintainability:

- This Characteristics of the software is important for both the software engineer and the user. If the change is to be required in the software, then that change leads to the change in the software so that it performs in accordance with the user requirement.
- The software engineer has to respond very fast if there is any change in the user requirements. Changes should be performed like that it will not affect the overall integrity of the software.

f. Portability:

A set of attributes that reflects on the ability of software to be transferred from one environment to another, without and minimum changes. The tasks performed are:

- Adaptability
- Install ability
- Replaceability

g. On-time:

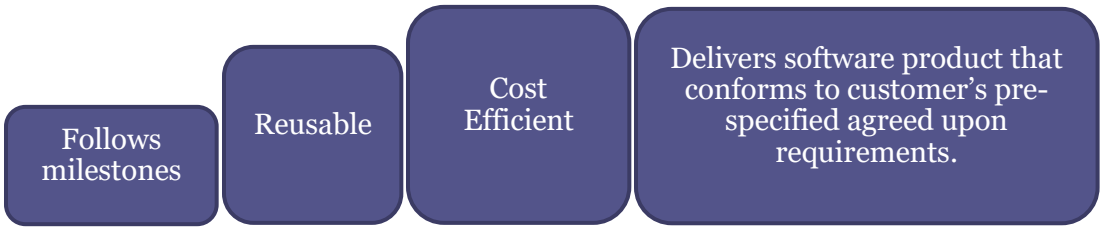
The software should be developed on-time. If the software is developed late then it is of no use because requirements change with the due time. A good engineer always develops the software on-time.

Apart from above mention qualities of software, there are various characteristics of software in software engineering:

h. Software is developed or engineered; it is not manufactured in the classical sense:

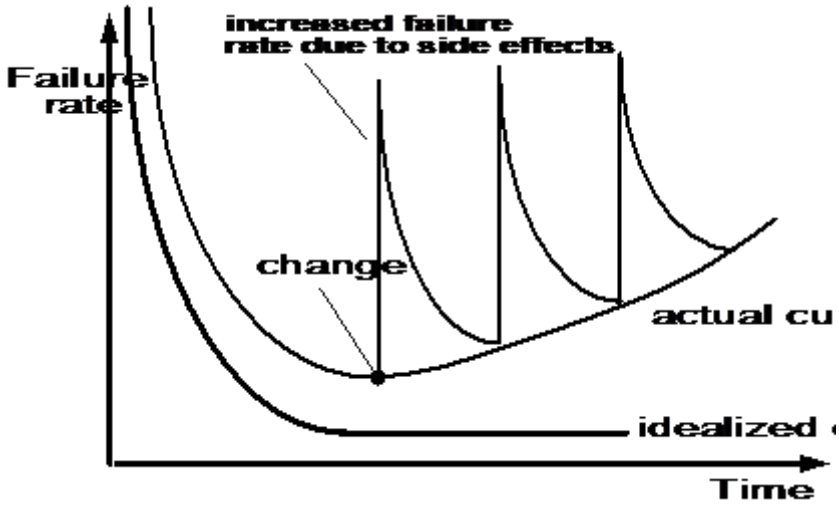
- Unlike hardware, software is logical rather than physical. It has to be designed well before producing it.
- Although some similarities exist between software development and hardware manufacturing, few activities are different. In both activities high quality is achieved through good design, but the manufacturing phase for hardware introduces quality problems that are easily correctable for software.
- Both activities require the construction of "product" but the approaches are different.

- **<u>In the Classical Sense, Software Engineering follows:</u>**



i. Software does not "wear out":

- ✓ As time progresses, the hardware components start deterioration-they are subjected to environmental problems such as dust, vibration, temperature etc. and at some point, of time they tend to break down.
- ✓ In software defects are corrected and the failure rate drops to a steady state level (hopefully quite low) for some period of time.
  - ✓ As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibrations, abuse, temperature extremes, etc. Stated simply, hardware begins to wear out.



1.6 Software Applications

Software may be applied in any situation for which a prespecified set of procedural steps (i.e., an algorithm) has been defined (notable exceptions to this rule are expert system software and neural network software). Information content and determinacy are important factors in determining the nature of a software application. Content refers to the meaning of incoming and outgoing information. For example, many business applications use highly structured input data (a database) and produce formatted "reports." Software that controls an automated machine (e.g., a numerical control) accepts discrete data items with limited structure and produces individual machine commands in rapid succession.

Information determinacy refers to the predictability of the order and timing of information. An engineering analysis program accepts data that have a predefined order, executes the analysis algorithm(s) without interruption, and produces resultant data in report or graphical format. Such applications are determinate. A multiuser operating system, on the other hand, accepts inputs that have varied content and arbitrary timing, executes algorithms that can be interrupted by external conditions, and produces output that varies as a function of environment and time. Applications with these characteristics are indeterminate.

It is somewhat difficult to develop meaningful generic categories for software applications. As software complexity grows, neat compartmentalization disappears. The following software areas indicate the breadth of potential applications:

a. **System software**. System software is a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) process complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, telecommunication processors) process largely indeterminate data. In either case, the system software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires

scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces.

b. Networking and Web Applications Software –

Networking Software provides the required support necessary for computers to interact with each other and with data storage facilities. The networking software is also used when software is running on a network of computers (such as the World Wide Web). It includes all network management software, server software, security and encryption software, and software to develop web-based applications like HTML, PHP, XML, etc.

c. Real-time software. Software that monitors/analyses/controls real-world events as they occur is called real-time. Elements of real-time software include a data gathering component that collects and formats information from an external environment, an analysis component that transforms information as required by the application, a control/output component that responds to the external environment, and a monitoring component that coordinates all other components so that real-time response (typically ranging from 1 millisecond to 1 second) can be maintained.

d. Business software. Business information processing is the largest single software application area. Discrete "systems" (e.g., payroll, accounts receivable/payable, inventory) have evolved into management information system (MIS) software that accesses one or more large databases containing business information. Applications in this area restructure existing data in a way that facilitates business operations or management decision-making. In addition to conventional data processing applications, business software applications also encompass interactive computing (e.g., point-of-sale transaction processing).

e. Engineering and scientific software. Engineering and scientific software have been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

f. Embedded software. Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. Embedded software can perform very limited and esoteric functions (e.g., keypad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

g. Personal computer software. The personal computer software market has burgeoned over the past two decades. Word processing, spreadsheets, computer

graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.

h. Utility Software –

The programs coming under this category perform specific tasks and are different from other software in terms of size, cost, and complexity. Examples are anti-virus software, voice recognition software, compression programs, etc.

i. Entertainment Software –

Education and entertainment software provides a powerful tool for educational agencies, especially those that deal with educating young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.

j. Document Management Software –

Document Management Software is used to track, manage and store documents in order to reduce the paperwork. Such systems are capable of keeping a record of the various versions created and modified by different users (history tracking). They commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities.

k. Web-based software. The Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats). In essence, the network becomes a massive computer providing an almost unlimited software resource that can be accessed by anyone with a modem.

l. Artificial intelligence software. Artificial intelligence (AI) software makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Expert systems, also called knowledge-based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game-playing is representative of applications within this category

1.7 Principles of Software Engineering

Software Engineering is the systematic engineering approach for software product/application development. It is an engineering branch associated with analysing user requirements, design, development, testing, and maintenance of software products.

Some basic principles of good software engineering are –

- **PRINCIPLE 1: BETTER REQUIREMENT ANALYSIS:** It is a fundamental software engineering technique that provides a comprehensive picture of the project. Hence user requirements add value to consumers by producing a high-quality software solution that satisfies their needs.

- **PRINCIPLE2:KISS**: All designs and implementations should be as basic as feasible, which implies adhering to the KISS (Keep it Simple, Stupid) philosophy. It simplifies code so that debugging and maintenance are simplified.

- **PRINCIPLE 3: CLEAR VISION**: The most crucial aspect of a software project's success is maintaining the project's vision throughout the development process. As a result of having a clear vision for the project, it can be developed properly.

- **PRINCIPLE 4: MODULARITY**: All functionalities should be designed using a modular approach to make development quicker and simpler. The principle of modularity is a specialization of the principle of separation of concerns. Following the principle of modularity implies separating software into components according to functionality and responsibility.

- **PRINCIPLE 5: ABSTRACTION:** Abstraction is a specialization of the idea of separation of concerns for hiding complicated things and offering simplicity to the customer/user, which implies it offers the user just what they need without knowing how it works.

- **PRINCIPLE 6: PROPER PLANNING**: Think before you act is a must-have concept in software engineering, which indicates that before you start implementing functionality, you must first consider application architecture since proper planning on the flow of project development yields better results.

- **PRINCIPLE 7: THE PRINCIPLE OF GENERALITY:** It means that it should not be limited or restricted to a specific set of cases/functions, but rather should be free from constraints and capable of providing comprehensive service to customers who have specific or general needs.

- **PRINCIPLE8: THE PRINCIPLE OF CONSISTENCY:** It is significant in coding style and GUI (Graphical User Interface) design because consistent coding style makes code simpler to understand and consistent GUI makes user learning of the interface and program easier.

- **PRINCIPLE 9: CONTINUOUS VALIDATION:** It ensures that a software system satisfies its requirements and serves its intended function, resulting in improved software quality control.

- **PRINCIPLE10: ANTICIPATION OF CHANGE:** The principle of anticipation of change recognizes the complexity of the learning process for both software developers and their clients. Preliminary requirements need to be worked out early, but it should be possible to make changes in the requirements as learning progresses.

- **Coupling** is a major obstacle to change. If two components are strongly coupled then it is likely that changing one will not work without changing the other.

- **Cohesiveness** has a positive effect on ease of change. Cohesive components are easier to reuse when requirements change.

Goal of software engineering:

The primary goals of software engineering are:

- To improve the quality of the software products.
- To increase the productivity.
- To give job satisfaction to the software engineers.

Software Requirements:

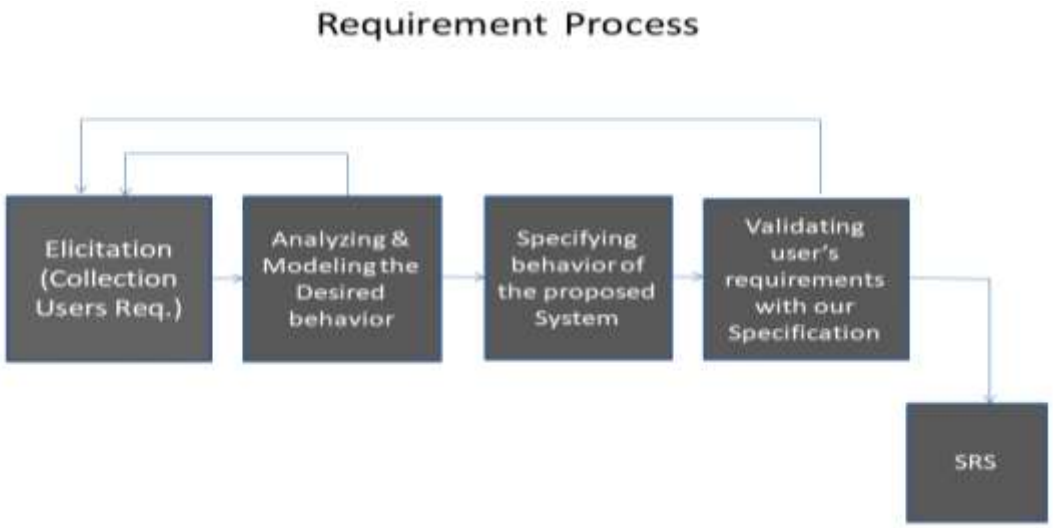
- A Requirement is a feature of the system or a description of something the system is capable of doing in order to fulfil the system's purpose.
- Requirements are descriptions of the services that a software system must provide and the constraints under which it must operate.

Types of Requirements Documents:

Requirement Definition is a document which is written in terms that the customer can understand. It includes all the aspects which are to be included in the product. It is prepared jointly by the Customers & Developers.

Requirement Specification restates what is defined in requirements definition in technical form. It is prepared for the development of system design. It is written by Requirement Analyst.

Requirement Process:



1.8 SRS (Software Requirement Specification)

An SRS is basically an organization's understanding (in writing) of a customer or client's system requirements and dependencies at a particular point in time prior to any actual design or development work. SRS is a description of the purpose of the proposed software without involving the description of how it will be done.

A software requirements specification (SRS) is a description of a software system to be developed, laying out functional and non-functional requirements, and may include a set of use cases that describe interactions the users will have with the software.

It establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do.

Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules.

The SRS document enlists enough and necessary requirements that are required for the project development. To derive the requirements, we need to have a clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of software.

1.8.1 Components of SRS:

The basic components an SRS must has

- ✓ Functional requirements
- ✓ Performance requirements
- ✓ Design constraints
- ✓ External Interface requirements.

1. Functional Requirements:

Functional requirements specify what output should be produced from the given inputs. So, they basically describe the connectivity between the input and output of the system. For each functional requirement:

1. A detailed description of all the data inputs and their sources, the units of measure and the range of valid inputs be specified.
2. All the operations to be performed on the input data to obtain the output should be specified.
3. Care must be taken not to specify any algorithms that are not parts of the system but that may be needed to implement the system.
4. It must clearly state what the system should do if system behaves abnormally when any invalid input is given or due to some error during computation.

5. Specifically, it should specify the behaviour of the system for invalid inputs and invalid outputs.

2. Non-Functional Requirements:

It deals with the Characteristics of the system which cannot be expressed as functions. Such as Portability, speed, size etc.

3. Customer Requirements:

It defines: -

- ✓ Where will the system be used?
- ✓ How will the system accomplish its mission objective?
- ✓ What are the critical system parameters to accomplish the mission?

4. Performance Requirements (Speed Requirements):

This part of an SRS specifies the performance constraints on the system software. All the requirements related to the performance characteristics of the system must be clearly specified. Performance requirements are typically expressed as processed transactions per second or response time from the system or screen refresh time or combination of these. It is a good idea to pin down performance requirements for the most used or critical transactions, user events and screens.

5. Interface Specification Requirements:

It defines interface, data structure and representation of data to be used.

6. Allocated Requirements:

It is established by dividing or allocating a higher-level requirement into multiple lower-level requirements.

7. Design Constraints:

The client environment may restrict the designer to include some design constraints that must be followed. The various design constraints are standard compliance, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

A. Standard Compliance: It specifies the requirements for the standard the system. The standards may include the report format and according procedures.

B. Hardware Limitations: The software needs some existing or predetermined hardware to operate, thus imposing restrictions on the design. Hardware limitations can include the types of machines to be used operating system availability, memory space etc.

C. Fault Tolerance: Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive, so they should be minimized.

D. Security: Currently security requirements have become essential and major for all types of systems. Security requirements place restrictions on the use of certain commands control access to database, provide different kinds of access, requirements for different people, require the use of passwords and cryptography techniques and maintain a log of activities in the system.

8. External Interface Requirements:

For each external interface requirements:

1. All the possible interactions of the software with people hardware and other software should be clearly specified.

2. The characteristics of each user interface of the software product should be specified.

3. The SRS should specify the logical characteristics of each interface between the software product and the hardware components for hardware interfacing.

I. SRS is a contract/ deed between customer and developer of system. SRS is systematic approach to formally the different binds of users, technical and development required for the deep analysis of

a) Present and future problems

b) Associated project development processes

c) Respective reliable solutions.

II. It is recorded in formal language or in graphical terms based on set deserve interfaces.

1.8.2 Activities of SRS:

a. Requirement Gathering:

The goals are:

- ✓ Studying existing documentation
- ✓ Interviewing different users
- ✓ Task Analysis (Book Issue, Return, Referrals)
- ✓ Scenario Analysis (Maximum number of books, Reference books reserved, service should be satisfactory)

b. Requirement analysis:

- ✓ Various alternative of problem solving
- ✓ Various I/O complexities that may arise.

c. Environmental Factors:

- ✓ Government policies
- ✓ Standards

- ✓ External Software

1.8.3 General Structure of SRS Document

Basic SRS Outline is as follows: -

Introduction

- Purpose
- Document Conventions
- Intended Audience
- Additional Information
- Contact Information of Team members
- References

Overall Description

- Product Functions
- User classes and characteristics
- Operating Environment

External interface Requirements

- User Interfaces
- Hardware Interfaces
- Software Interfaces
- Communication Protocols and interfaces.

System Features

- Description and Priority
- Action/Result
- Functional requirements

Other Non-Functional Requirements

- Performance Requirements
- Safety Requirements
- Security Requirements
- Software Quality Attributes
- Project Documentation
- User documentation

Other Requirements

1.8.4 Goals/ Advantages of SRS:

1) It establishes session between users because 50 percent of the problems arise due to non-specification of unknown requirements and unmentioned resources.
2) Deep analysis of customer and management requirements cover various kinds of risks, future delays and over budgets.
3) Behaviour of system towards environment.
4) It provides feedback to the customer.
5) It decomposes the problem into component parts.
6) It serves as input to the design specification.
7) It serves as a product validation check

Example: The SRS may be one of a contract deliverable data item descriptions or have other forms of organization. An example organization of an SRS is as follows.

- ✓ **Introduction and Purpose**
  - Definitions and System Overview
  - References
- ✓ **Overall Description**
  - Product perspective
  - System and User Interfaces
  - Hardware and Software Interfaces
  - Communication Interfaces and Memory Constraints
  - Operations and Site Adaptation Requirements
  - Product functions and User characteristics
  - Constraints, assumptions and dependencies.
- ✓ **Specific requirements**
  - External interface requirements
  - Functional requirements
  - Performance requirements
  - Design constraints and Standards Compliance
- ✓ **Logical database requirement**
- ✓ **Software system attributes**
  - Reliability

- Availability
- Security
- Maintainability
- Portability

✓ **Other requirements**

1.9 Metrics for Software requirement specification quality

The notation SRS, which is signified as software requirements specifications document reports all functionalities and abilities that a software has to provide and states any pre-requisite restrict through which the system has to stand.

It is made up of two terms, **the term requirement is stated as an objective, which one has to be met**, whereas the specification **explains how this objective would be attained**.

On the other hand, the specification document explains how particular tasks have to be done. Here, proactive involvement acts as a major role in quality assurance at the time of the system's requirements specification stage.

For measuring the quality of the SDLC process, the list of quality features, which are expected the software requirements specifications, has to be compiled. Various metrics are given as:

1. Completeness The metric completeness denotes the description for every requirement labelled in the first order of the SRS template. However, having a description for every requirement is not enough to state the completeness. Since the description of a requirement shall include details related to functional and non-functional factors, properties of the inputs, and outcomes of the corresponding requirement.

2. Correctness:

Every requirement shall represent something required of the system to be built. However, the accuracy of this metric value is proportionate to the SRS quality estimation skills of the human resource, which is highly vulnerable to the false alarming of SRS correctness assessment. The ratio of requirements reflecting the inputs and outcomes fall under the business rules defined for target software has considered estimating the SRS's correctness.

3. Unambiguity:

Every requirement denotes one interpretation (An SRS is unambiguous if and only if every requirement stated if it has only one possible interpretation). However, more than one interpretation of the requirement appears for different inputs or for different modules. Hence, the unambiguity has modularized as the ratio of requirements having a unique interpretation and having multiple interpretations with different formats of input.

4. Understandability

The traditional definition of metric understandability is the description of a requirement that shall fall into the scope of all classes of people related to the target software. However, it is

impractical to justify the perception of different classes of people related to that requirement. Hence, the understandability has modularized based on completeness and correctness, which are understandable.

5. Traceability:

Documenting factors are critical to trace the requirements and corresponding descriptions. The requirements and corresponding descriptions well referred in baselined documents include system-level requirements specifications, statements of work (SOW), white papers and an earlier version of the SRS to which this new SRS must be upward compatible. Hence the estimation of traceability has modularized as the description of system-level requirements.

6. Modifiability:

The requirements having dependents and requirements dependent on other requirements are complexed to modify. However, the dependency of specific operable sources also detracts from the scope of modifiability. Hence, the estimation of the metric modifiability is modularized as not dependent on specific operable sources.

7. Verifiability:

Verifiability is poor if the descriptions are ambiguous, not worth to cost (ratio of excess cost), no sources to test, not possible to test, poor template measures, and fault references. The metric verifiability depends on two other metrics: non-ambiguous and traceability.

Thus, the scope to perform testing and the availability of sources to perform testing improve the requirements' verifiability as:

(i) Unambiguity of the SRS,

(ii) Traceability of the SRS,

(iii) The ratio of requirements having scope to perform testing

(iv) The ratio of requirements having sources to perform testing.

8. Consistency:

The consistency of a requirement denotes no conflicts between the statements stated in the corresponding requirement.

The overall consistency of the software requirements specification has modularized the ratio of requirements not conflicting with earlier versions of the SRS, and the ratio of requirements stated in white papers with no conflicts.

1.10 Summary

Software engineering is a practical implementation of various scientific and engineering principles to develop mega projects. Various software attributes and principles to be adhere with are specified to deliver the efficient project in time and budget.

1.11 Suggested Readings

1. "Software Engineering: A Practitioner's Approach" 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.

2. “Software Engineering” by Ian Sommerville, Addison-Wesley, 7th Edition.
3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

1.12 Model Questions

1. Define software and its various characteristics in detail.
2. Explain various Software engineering principles.

Lesson-2

SOFTWARE PROCESS MODELS

Structure

2.0 Objectives

SDLC has three primary business objectives:

- learners will learn about the delivery of high-quality systems through SDLC.
- learners will understand the role of SDLC in providing strong management controls.
- learners will appreciate the role of SDLC in maximizing productivity.

2.1 Introduction

To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods, and the generic phases. This strategy is often referred to as a process model or a software engineering paradigm.

A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software. The SDLC aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

SDLC is the acronym of Software Development Life Cycle.

- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes.
- It aims to be the standard that defines all the tasks required for developing and maintaining software.

2.2 Software development life cycle (SDLC)

Software development life cycle (SDLC) is also referred to as the application development life cycle. It is a term used in system engineering and software engineering to describe the process for planning, developing, testing, and deploying an information systems. SDLC is a life cycle through which software goes, till it is fully developed and deployed.

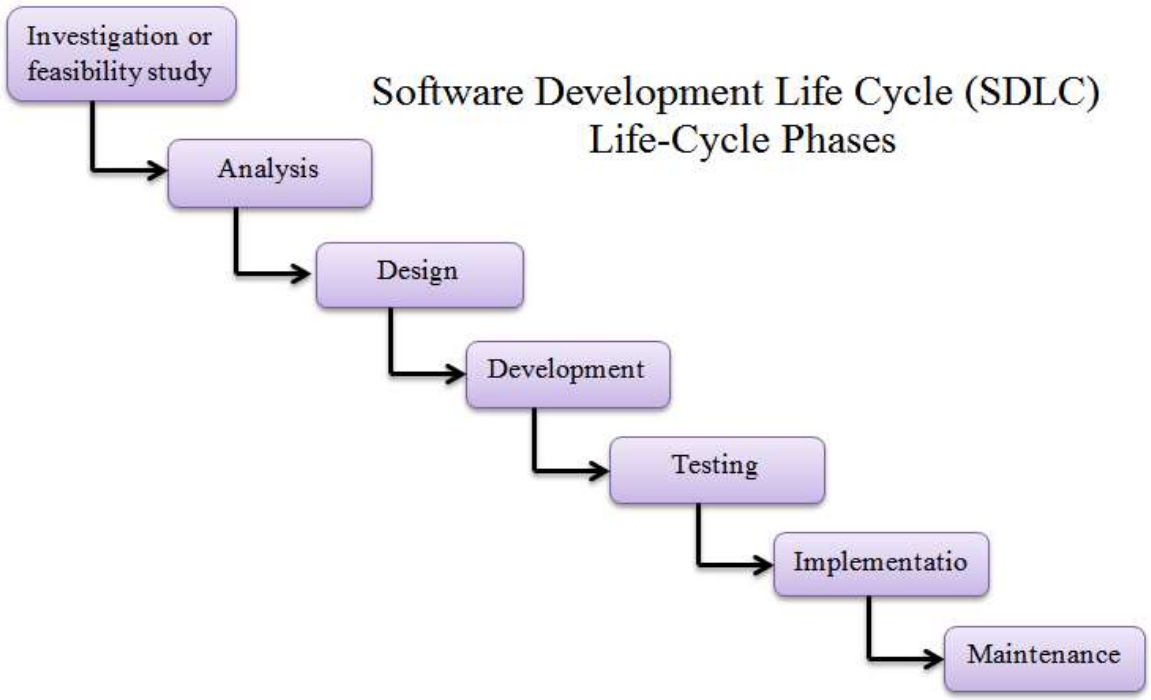


Fig1: Phases of SDLC

The Phases of SDLC are described as below:

1. Feasibility Study or Requirement Analysis:

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

2. Design:

It has two steps:

(a). High level design (HLD): It gives the architecture of software product.

(b). Low level design (LLD): It describes how each and every feature in the product should work and every component.

3. Development:

This is the longest phase.

This phase consists of Front end + Middle ware + Back-end

In front end: development coding is done even SEO setting is done

In Middle ware: They connect both front end and back end

In back-end: database is created.

4. Testing:

Testing is carried out to verify the entire system. The aim of the tester is to find out the gaps and defects within the system and also to check whether the system is running according to the requirement of the customer/client.

5. Implementation:

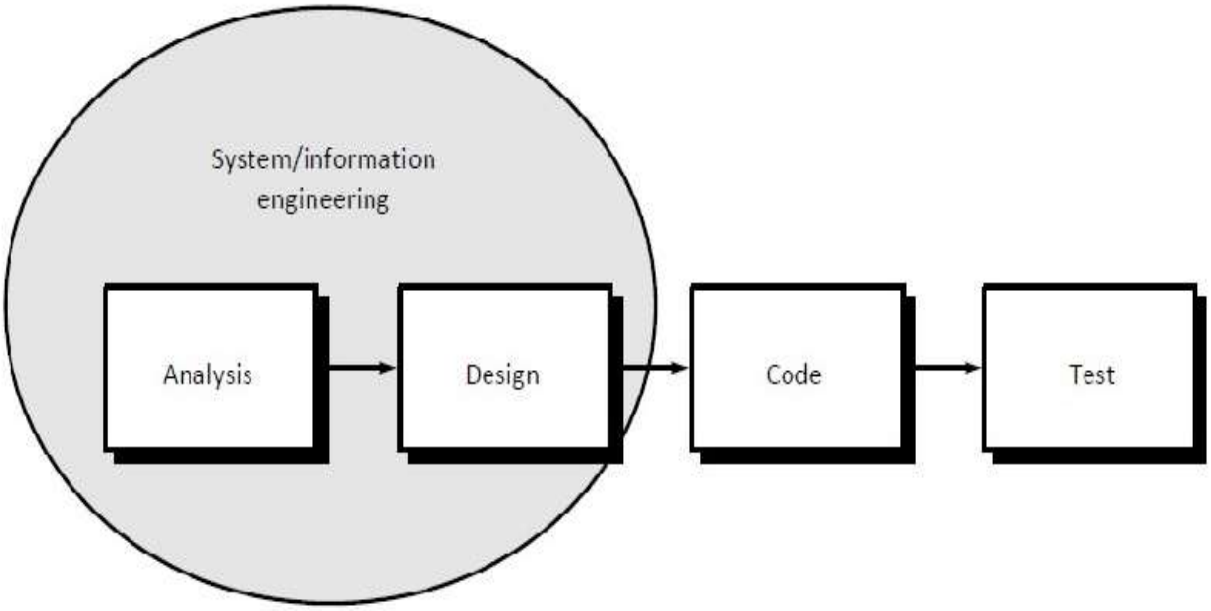
After successful testing the product is delivered/deployed to the client, even client is trained how to use the product**.**

6. Maintenance:

Once the product has been delivered to the client a task of maintenance starts as when the client will come up with an error the issue should be fixed from time to time.

2.3 Waterfall Model

The waterfall model also called as the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.



The Waterfall model encompasses the following activities:

System/information engineering and modelling:

Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when software must interact with other elements such as hardware, people, and databases.

a. System engineering and analysis encompass requirements gathering at the system level with a small amount of top-level design and analysis.

b. Information engineering encompasses requirements gathering at the strategic business level and at the business area level.

Step 1. Software requirements analysis:

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

Step 2. Design:

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

Step 3. Code generation:

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

Step 4. Testing:

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

Step 5. Support:

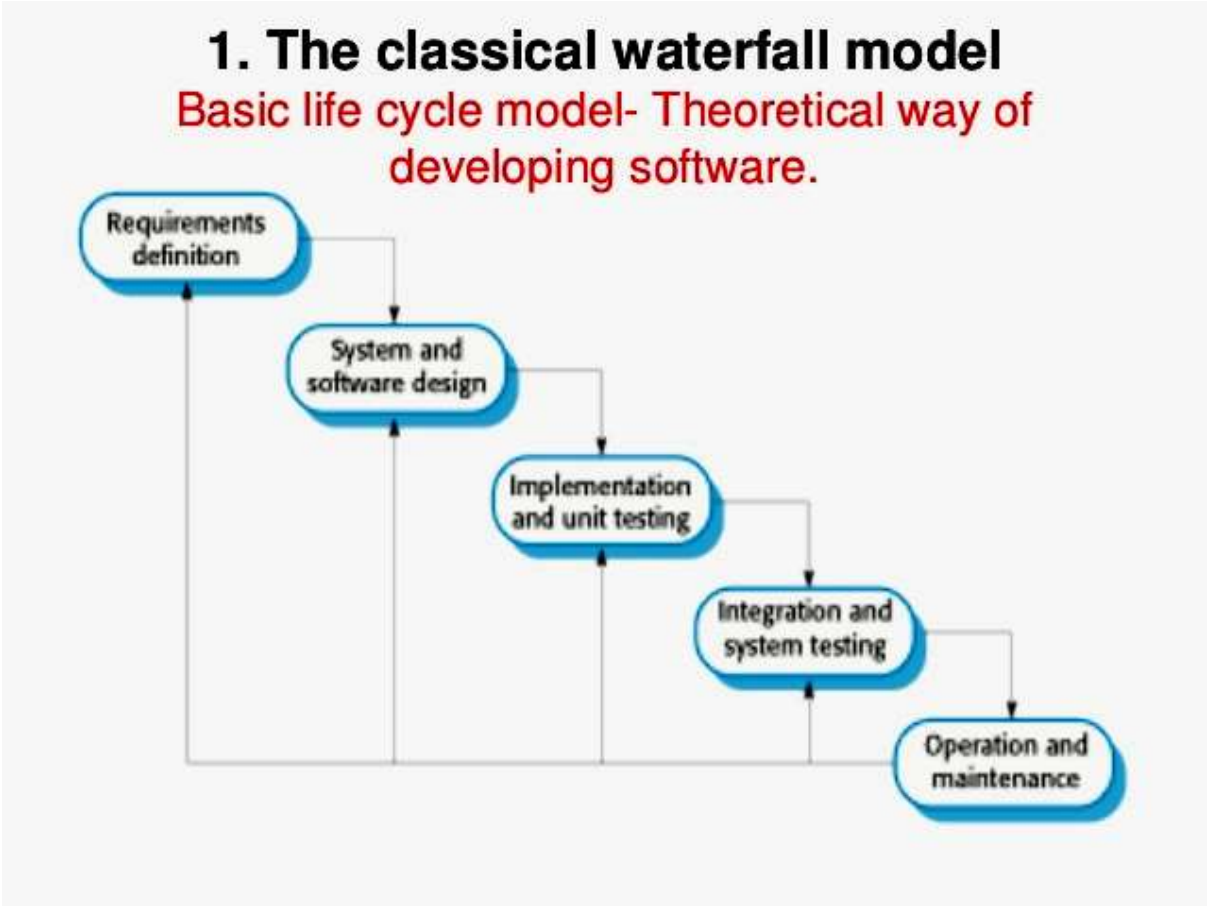
Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software). Change will occur because errors have been encountered because the software must be adapted to accommodate changes in its external environment. For example, a change required because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

WATERFALL MODEL PROBLEMS:

- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and at the early stage
- Requires customer patience because a working version of the program doesn't occur until the final phase.

Features:

- The waterfall method has generally used by software engineers for all but only for the simplest projects and it is inflexible to deal with the complex and rapidly changing nature of software.
- Development moves from concept, through design, implementation, testing, installation, trouble-shooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order, without any overlapping or iterative steps.



Advantages:

a. The advantage of waterfall development is that it allows for departmentalization and managerial control.

b. As a result, a schedule can be set with deadlines.

Disadvantages:

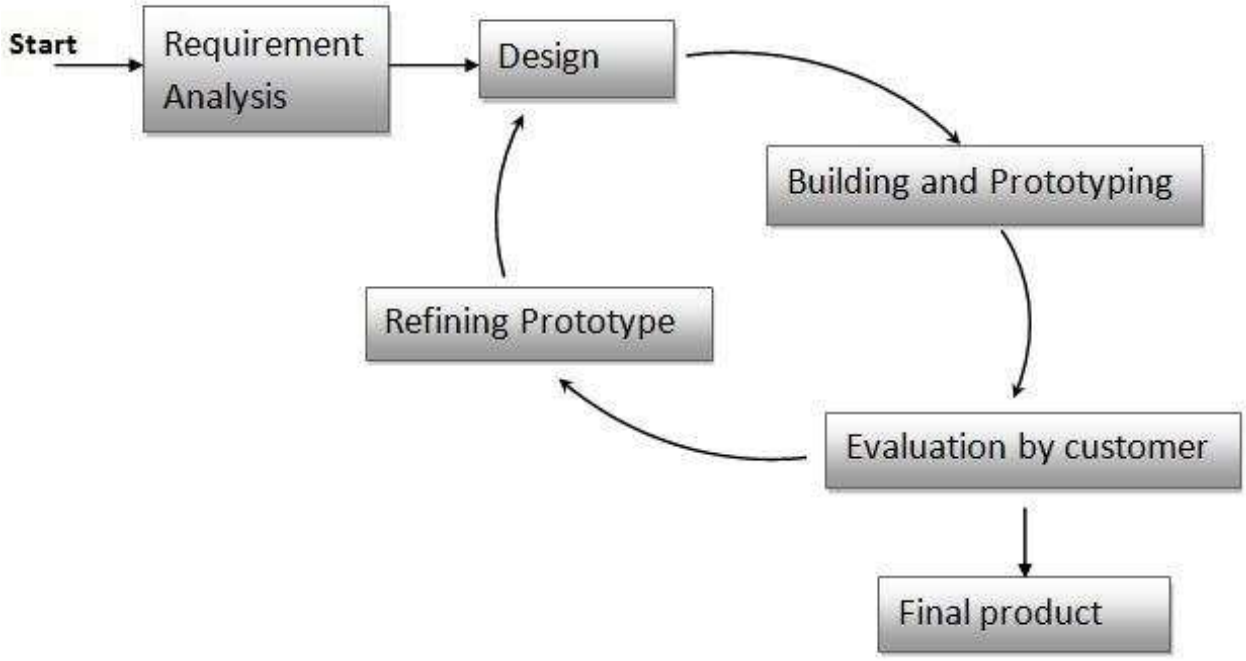
a. The next development phase proceeds only when previous phase of development is completed and there is no turning back.

b. The disadvantage of waterfall development is that it does not allow for much reflection or revision.

c. Once an application is in the testing stage, it is very difficult to go back and change some-thing that was not well defined in requirement phase.

2.4 Prototyping model

The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.



In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle.

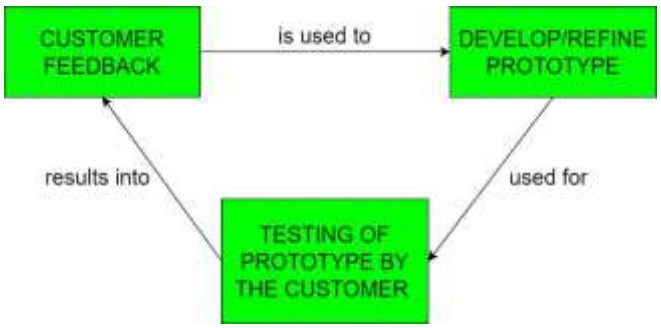
a. The process starts by interviewing the customers and developing the incomplete high-level paper model.

b. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer.

c. Once the customer figures out the problems, the prototype is further refined to eliminate them.

d. The process continues until the user approves the prototype and finds the working model to be satisfactory**.**

Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small-scale replica of the end product and is used for obtaining customer feedback as described below:



There are four types of models available:

A) Rapid Throwaway Prototyping –

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality**.**

B) Evolutionary Prototyping –

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers**.**

C) Incremental Prototyping – In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It's a very efficient approach which reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually. The time interval between the project begins and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously.

D) Extreme Prototyping – This method is mainly used for web development. It is consisting of three sequential independent phases:

D.1) In this phase a basic prototype with all the existing static pages is presented in the HTML format.

D.2) In the second phase, Functional screens are made with a simulate data process using a prototype services layer.

D.3) This is the final step where all the services are implemented and associated with the final prototype.

This Extreme Prototyping method makes the project cycling and delivery robust and fast, and keeps the entire developer team focus centralized on products deliveries rather than discovering all possible needs and specifications and adding unnecessitated features.

Advantages –

1. The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
2. New requirements can be easily accommodated as there is scope for refinement.
3. Missing functionalities can be easily figured out.
4. Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
5. The developed prototype can be reused by the developer for more complicated projects in the future.
6. Flexibility in design.

Disadvantages –

1. Costly w.r.t time as well as money.
2. There may be too much variation in requirements each time the prototype is evaluated by the customer.
3. Poor Documentation due to continuously changing customer requirements.
4. It is very difficult for developers to accommodate all the changes demanded by the customer.
5. There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
6. After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
7. The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

Uses –

1. The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable.
2. It can also be used if requirements are changing quickly.
3. This model can be successfully used for developing user interfaces, high technology software-intensive systems, and systems with complex algorithms and interfaces.
4. It is also a very good choice to demonstrate the technical feasibility of the product.

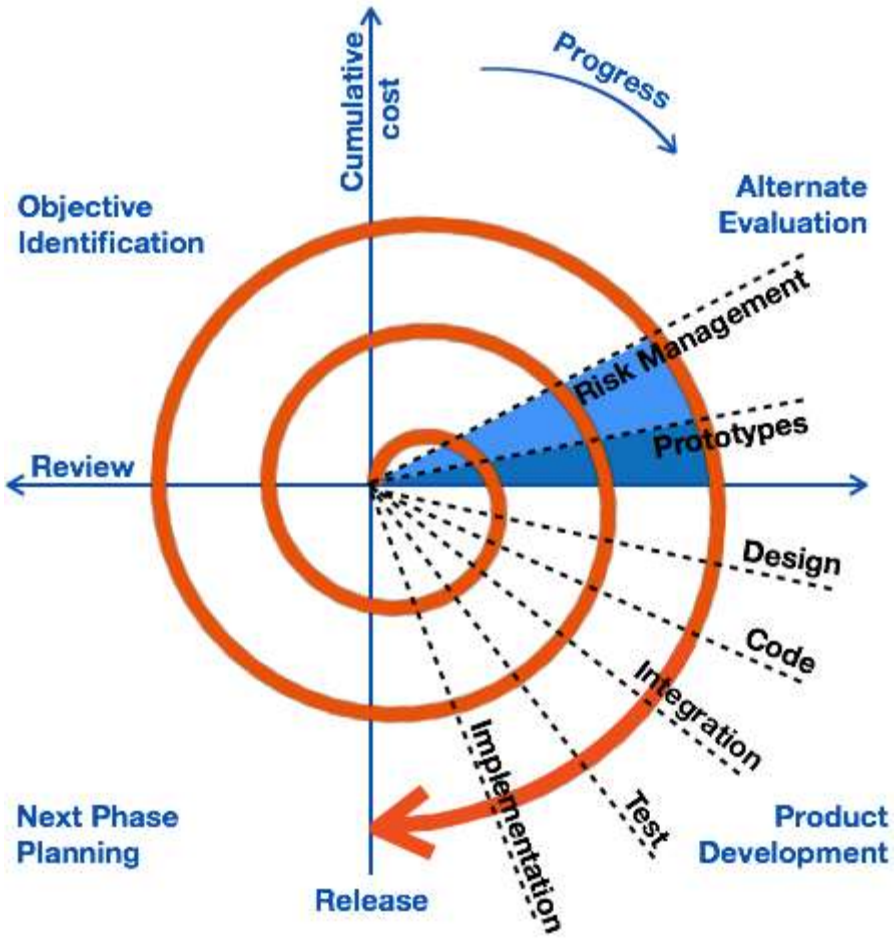
2.5 Spiral Model

The spiral model, originally proposed by Dr. Barry Boehm in 1988. It is an evolutionary software process model that couples the iterative nature of prototyping with the

controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions.

Typically, there are between three and six task regions.



- **Customer communication**—tasks required to establish effective communication between developer and customer.
- **Planning**—tasks required to define resources, timelines, and other project related information
- **Risk analysis**—tasks required to assess both technical and management risks.
- **Engineering**—tasks required to build one or more representations of the application.

- **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).
- **Customer evaluation**—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

Features:

The specification, development and validated continues for the next stage until it meets the final specification and is accepted by the customer as complete.

This approach ensures, through wide usage of risk handling, that the product fits the specification.

2.6 The WINWIN Spiral Model

Boehm's WINWIN spiral model developed by Boehm in 1998 that defines a set of negotiation activities at the beginning of each pass around the spiral. Rather than a single customer communication activity, the following activities are defined:

1. Identification of the system or subsystem's key "stakeholders."
2. Determination of the stakeholders' "win conditions."
3. Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned (including the software project team).

Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to software and system definition.

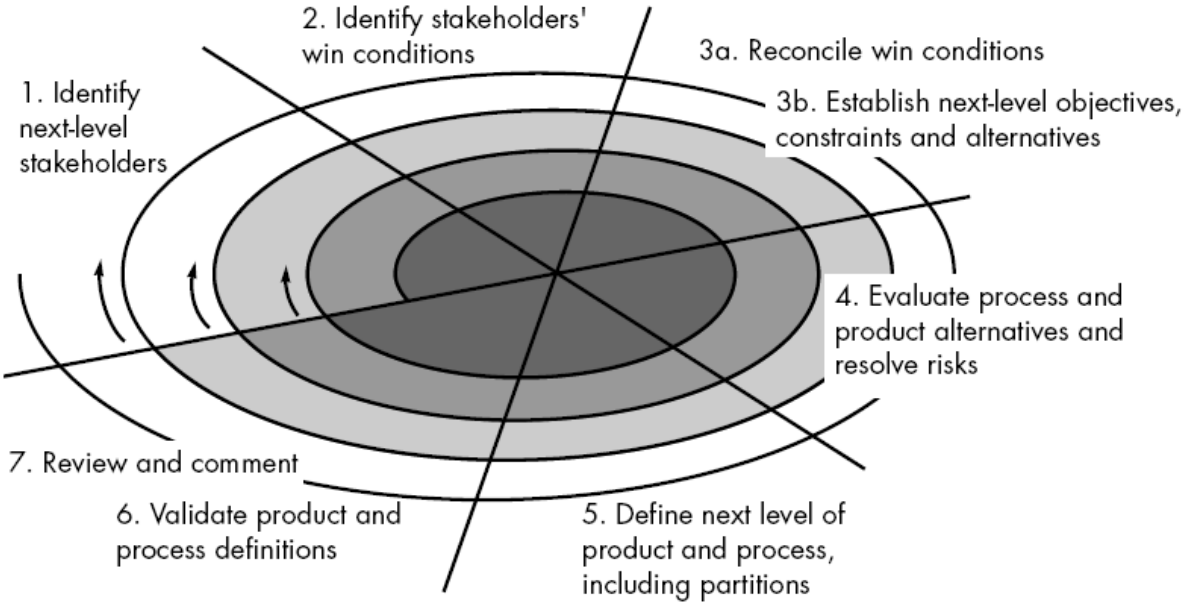


Fig: The WINWIN spiral model

2.7 Agile Model

Agile SDLC model is a combination of iterative and incremental process models which focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like –

- ✓ Planning
- ✓ Requirements Analysis
- ✓ Design
- ✓ Coding
- ✓ Unit Testing and
- ✓ Acceptance Testing.

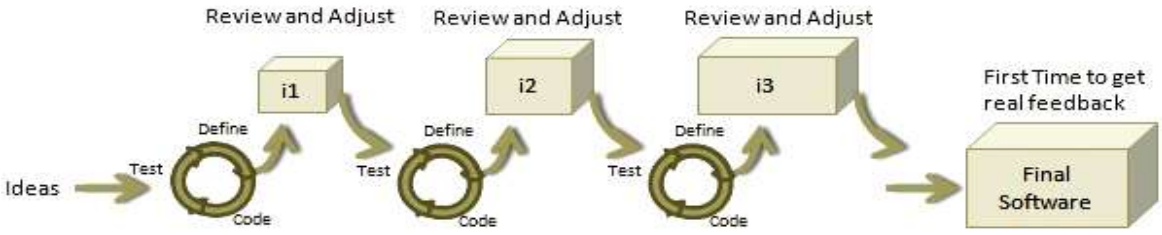
At the end of the iteration, a working product is displayed to the customer and important stakeholders.

Agile Definition:

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to small time frames to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Traditional Method



Agile Method

The most popular Agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.

Twelve Principles of Agile Manifesto

- **Customer Satisfaction** − Highest priority is given to satisfy the requirements of customers through early and continuous delivery of valuable software.
- **Welcome Change** − Changes are inevitable during software development. Ever-changing requirements should be welcome, even late in the development phase.
- **Deliver a Working Software** − Deliver a working software frequently, ranging from a few weeks to a few months.
- **Collaboration** − Business people and developers must work together during the entire life of a project.
- **Motivation** − Projects should be built around motivated individuals. Provide an environment to support individual team members and trust them so as to make them feel responsible to get the job done.
- **Face-to-face Conversation** − Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team.
- **Measure the Progress as per the Working Software** − Working software is the key and it should be the primary measure of progress.
- **Maintain Constant Pace** − Agile processes aim towards sustainable development. The business, the developers, and the users should be able to maintain a constant pace with the project.
- **Monitoring** − Pay regular attention to technical excellence and good design to enhance agility.
- **Simplicity** − Keep things simple and use simple terms to measure the work that is not completed.
- **Self-organized Teams** − An agile team should be self-organized and should not depend heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams.
- **Review the Work Regularly** − Review the work done at regular intervals so that the team can reflect on how to become more effective and adjust its behaviour accordingly.

Iterative/incremental and Ready to Evolve

1. Most of the agile development methods break a problem into smaller tasks. There is no direct long-term planning for any requirement. Normally, iterations are planned which are for very short period of time, for example, 1 to 4 weeks.

2. A cross-functional team is created for each iteration that works in all functions of software development like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
3. The result at the end of the iteration is a working product and it is demonstrated to the stakeholders at the end of an iteration.
4. An information radiator (physical display) is normally located prominently in an office, where passers-by can see the progress of the agile team. This information radiator shows an up-to-date summary of the status of a project.

Agile Vs Traditional SDLC Models

1. Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach.
2. Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed whereas Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle.
3. In Agile Models, there is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future but not in case of traditional SDLC models.

Agile Model - Pros and Cons:

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows −

a. It is a very realistic approach to software development.
b. Promotes teamwork and cross training.
c. Functionality can be developed rapidly and demonstrated.
d. Resource requirements are minimum.
e. Suitable for fixed or changing requirements
f. Delivers early partial working solutions.
g. Good model for environments that change steadily.
h. Minimal rules, documentation easily employed.
i. Enables concurrent development and delivery within an overall planned context.
j. Little or no planning required.
k. Easy to manage.

l. Gives flexibility to developers.

The disadvantages of the Agile Model are as follows –

a. Not suitable for handling complex dependencies.

b. More risk of sustainability, maintainability and extensibility.

c. An overall plan, an agile leader and agile PM practice is a must without which it will not work.

d. Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.

e. There is a very high individual dependency, since there is minimum documentation generated.

f. Transfer of technology to new team members may be quite challenging due to lack of documentation.

2.8 Summary

The success of any project depends on the selection and implementation of various process development models. The selection of particular method depends on the project size, type, nature and availability of resources.

2.9 Suggested Readings

1. "Software Engineering: A Practitioner's Approach" 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.

2. "Software Engineering" by Ian Sommerville, Addison-Wesley, 7th Edition.

3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

2.10 Model Questions

1. Explain SDLC in detail.
2. Compare Waterfall and Spiral models with examples.
3. Draw difference between Spiral and win –win spiral models.
4. Explain Agile Model in Details.

Lesson-3

SOFTWARE PROJECT MANAGEMENT

Structure

3.0 Objectives

Management is the process of completing the activities efficiently and effectively with and through the people. Project management is the art of matching a project to milestones, tasks, and resources to accomplish that goal as needed. "As needed" because one has limited time, money, and resources (human and machinery) with which to accomplish a goal. So, the project is a process of planning, execution, monitoring, and controlling.

3.1 Introduction

Software project management is required:

- To provide the basic skills and knowledge needed to effectively manage a group project.
- Breaking a complex project into manageable sub-projects
- Assigning responsibility and ownership of project components
- Project scheduling and understanding.
- On-time delivery

- To provide desired performance.
- Risk handling as risk is crucial for the timelysuccess event.
- Completing and executing the laid project plan

Figure 1 shows the purpose of project management

Figure 1: Project management

3.2 Software project: Definition

A project is a well-defined task, which is a collection of several operations performed 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 a routine activity or day-to-day operation.
- 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.

A software project must be completed within the stipulated time so it has specified a set of inputs and outputs to achieve a specific goal. For a better software project, there is a need for project managers and teammembers tosplit theproject into specific tasks so that they can manage their responsibilities and utilize theteam's strength in an efficient manner.

Project Manager

A project manager is a person who has the overall responsibility for the planning, design, execution, monitoring, controlling, and closure of a project. There are few responsibilities of a project managers:

Responsibilities of a Project Manager:

- Managing risks and issues.
- Creating a project team and assigning tasks to several team members.
- Activity planning and sequencing.
- Monitoring and reporting progress.
- Modifying the project plan to deal with the emerging situations.

3.3 Software Project Management:

Definition:

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software engineering in which software projects are planned, implemented, monitored, and controlled.

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

It is a procedure of managing, allocating, and timing resources to develop computer software that fulfils user requirements. In software project management, the clients and the developers need to know the length, period, and cost of the projectand the efficiency of software project management is validated through three crucial parameters of Time, Cost and Quality.

Fig 2: Project Management Parameters

It is an essential responsibility of the software organization to deliver a quality product, keeping the cost within the client's budget and deliver the project as per scheduled time. There are several factors, both internal and external, which may impact this triple constraint triangle. Any of the three factors can severely impact the other two. Like if the project developed to launch on 31st December is not validated and delivered before 31st December then this can cost the reputation and integrity of the organization against their competitors and in long term the organization may lose its prime customers.

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

3.3.1 Project Management Activities

Software project management comprise ofthe number of activities, like planning of the project, deciding the scope of the software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. These activities are broadly categorized as:

A. Project Planning

B. Scope Management

C. Project Estimation

A. Project Planning

Software project planning is a task, which is performed before the production of software actually starts. It is used for software production but involves no existing activity that has any direct connection with software production; rather it is a set of multiple processes, which facilitates software production. It includes:

- Identification of activities.
- Causes (reports, manual) for management.
- Deliverables for the customer.

B. Scope Management

It is a set of all scope related activities and processes that 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 limits the project to predefined quantifiable tasks, which can easily be documented in order to avoid 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

C. Software Project Estimation

For effective management and accurate estimation, there is a need for various measures. With correct estimation, managers can manage and control the project more efficiently and effectively to deliver it within the estimated and sanctioned budget.

3.3.2 The important Project estimation parameters are:

C.1 Software Size Estimation

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

C.2 Effort estimation

The managers estimate efforts in terms of number of persons and man-hours required to produce the software. For effort estimation software size should be known. This can either be derived from managers' experience and organization's historical data. Then this estimated software size can be converted into efforts by using some standard formulae like COCOMO model.

C.3 Time estimation

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

Here

Total time required to complete project= sum of time required to complete all n subtasks in hours/ days

C.4 Cost estimation

This is the most difficult aspect because it depends on multiple more factors than any of the previous ones. For estimating project cost, it is required to consider -

- Size of software
- Software quality
- Hardware
- Additional tools and licenses
- Skilled personnel with task-specific skills
- Travel involved
- Risks involved
- Communication
- Training and support

D. Project Estimation Techniques

The important parameters of project estimation are size, effort, time, and cost.

The project manager can calculate and quantify these factors using two broadly recognized the techniques like:

I. Decomposition Techniques

Software project estimation is a form of problem-solving that is, developing a cost and effort estimate for a software project. For this reason, the problem should be decomposed into a set of smaller problems. Thesetechniques assume the software as a product of various compositions. There are two main models -

- **Lines of Code (LOC)**: The size of the project is estimated according to the number of lines of codes in the software product but it is dependable on coding languages.
- **Function Points (FP):** The number of function points are calculated in the software product. It measures the different function modules required to deliver the user wished facilities.

II. Empirical Estimation Techniques

These techniques use empirically derived through well-implemented formulae to make estimations. These formulae are based on LOC and FPs. The various techniques are:

- **Putnam Model**

This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps the time and efforts required for the given 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 projects.

E. Project Scheduling

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

For scheduling a project, it is necessary to -

1. Find out various tasks and correlate them.
2. Break down the major project tasks into smaller and manageable modules.
3. Estimate the time frame required for each task
4. Divide time into sub work-units
5. Assign an adequate number of work units for each task
6. Calculate the total time required for the project from start to finish

3.4 Project Scheduling Techniques:

The risk and uncertainty rise with respect to the size of the project, even when the project is developed according to set methodologies. Hence these are various available

3.4.1 Gantt Chart

Gantt charts were devised by Henry Gantt (1917).

a. It is a type of chart in which a series of horizontal lines show the amount of work done or production completed in a given period of time in relation to the amount planned for those projects. It represents the project schedule with respect to time periods. Gantt chart is usually utilized in project management, and it is one of the most popular and helpful ways of showing activities displayed against time. Each activity is represented by a bar.

b. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities. It is simply used for graphical representation of a schedule that helps to plan in an efficient way, coordinate, and track some particular tasks in a project. Its various features are

    1. The purpose of the Gantt chart is to emphasize the scope of individual tasks. Hence set of tasks is given as input to the Gantt chart.
    2. Gantt chart is also known as timeline chart. It can be developed for the entire project or it can be developed for individual functions.
    3. After the generation of a timeline chart, project tables are prepared.
    4. In project tables list, all tasks in a proper manner along with the start date and end date and information related to it.

Importance of Gantt charts:

A Gantt chart is a useful tool when an organization requires the entire landscape of either one or multiple projects. It helps to view tasks that are dependent on one another as well as an event that is coming up.

Project managers use Gantt charts for three main reasons to:

1. Build and manage a comprehensive project.
2. Determine logistics and task dependencies.
3. Monitor progress of a project

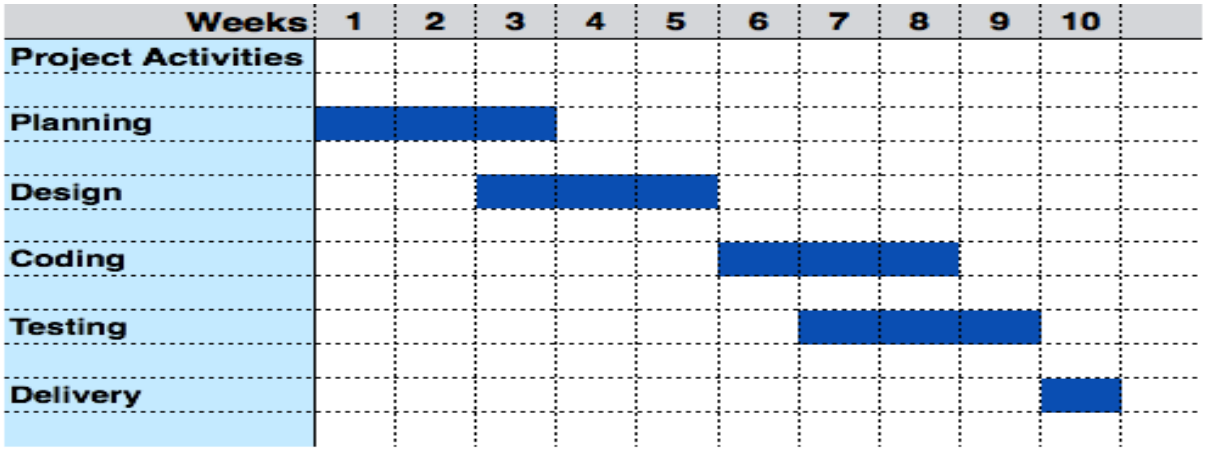


Fig 3: Gantt Charts

Advantages of GANTT Charts

- Shows slack times through dashed lines
- Critical path events are defined as milestones
- Can be used as a scheduling mechanism.

Limitations:

- Sometimes, Gantt chart makes the project more complex in case of overlapping of activities.
- The size of bar chart does not necessarily indicate the amount of work done in the project.
- Gantt charts and projects are needed to be updated on regular basis due to market changing scenario.

3.4.2 PERT Chart

PERT (Program Evaluation & Review Technique) chart is a tool with following features.

1. It depicts projects as a network diagram.
2. It is capable of graphically representing the main events of the project in both parallel and consecutive ways.
3. Events, which occur one after another, show dependency of the later event over the previous one.

Structure of PERT Chart should include following components.

Numbered nodes: These are the numbered boxes. Some project managers choose to draw them as circles and Each node represents an event or milestone in the project, completion of one stage, or a series of tasks needed to move the project forward.

a. **Directional arrows:** The arrows on a PERT chart represent the tasks or activities that need to be completed before the team can move on to the next event or phase in the

project. The task between nodes 1 and 2 and 5 and 6 are directional arrows and should be done in sequence from 1 to 2 and 5 to 6.

b. **Divergent arrows:** These arrows represent tasks that a team may work on simultaneously or in any sequence they choose because they do not have dependencies. As shown in the figure, the team may work first on the activities leading to node 3 or node 4, or they may choose to complete them simultaneously.

c.

Fig 4: PERT Chart

Steps to create a PERT chart, a project management team should follow these steps.

Step 1: Identify all of the project's activities.

First, define all of the major phases, milestones, and tasks needed to complete the project. Also, verify those milestones against user commitments.

Step 2: Identify dependencies

If you determine some tasks or activities have dependencies, you will want to depict those tasks with directional arrows. This will ensure your team knows the sequence they need to tackle each task.

Step 3: Draw your chart.

i. Take the events and milestones (numbered nodes) you've identified and draw them out.

ii. Write out the tasks and activities that the team must complete between each node.

iii. Draw directional arrows or divergent arrows accordingly to show the interdependencies.

Step 4: Establish timelines for all activities.

You should now set a timeframe when the team will need to complete those tasks along with all arrows. For example, in fig 5 activity "C" has a timeframe of 1 day. This represents the estimated timeframe and/or deadline you set for the activity.

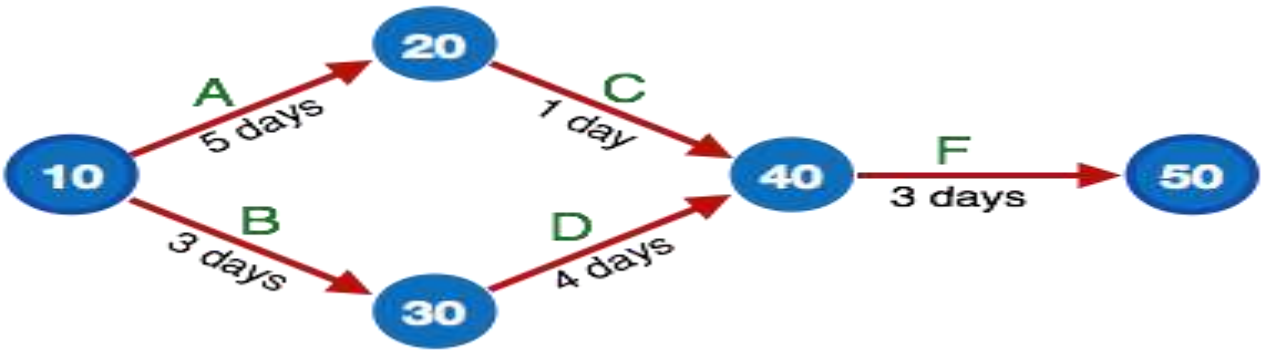


Fig 5: PERT chart example

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

In this technique, a numerical number is used to represent the duration of each activity.

Importance:

- PERT is mainly concerned with **the time of each activity and interrelations among activities**
- PERT is used in **R&D Development**
- It is originally focused on time and uses **probabilistic analysis**.
- The PERT technique provides the ability **to evaluate the time and resources necessary** for a project by **tracking required assets** at each stage of the process, as well as throughout the course of the project.
- PERT charts are useful in **what-if analyses**, helping companies understand all possible workflows and choose the most efficient and beneficial path.
- The analysis of the PERT chart includes **data from various departments** within an organization. Combining all of the information helps identify each responsible team within the company, while facilitating an environment where each department takes responsibility for its work.
- The process of creating a PERT chart also **improves communication** and enables an organization to invest energy in projects that will **enhance its strategic positioning**.
- PERT charts **make unclear deadlines more predictable**, clarify dependencies between tasks and establish a clear order for completing the tasks.
- **It implements Critical path & slack**.

Disadvantages of PERT charts:

- A strict focus on deadlines may deviate managers to estimate the full financial positioning of the project.
- PERT charts lack the flexibility to adapt to small changes that occur when confronted with a unseen market fluctuations.

- If any calculations are inaccurate in the creation of the chart, delays could occur, causing bottlenecks and negatively impacting the final delivery date.
- PERT charts are subjective; their success depends on the experience of the project manager. Consequently, some charts may include unreliable data or unrealistic expectations for the cost and time frame of the project.
- Creating a PERT chart is labour-intensive, requiring additional time and resources.
- In order for the chart to remain valuable, it must be consistently reviewed and maintained.

3.4.3 Critical Path Analysis/ Critical Path Method (CPM)

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 the 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.

1. The events are arranged according to their earliest possible start time.
2. 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. By drawing a network diagram of the underlying project one can figure out the critical path of your project. The critical path is the longest path through the network. If something falls behind schedule on the critical path, the whole project falls behind the schedule.

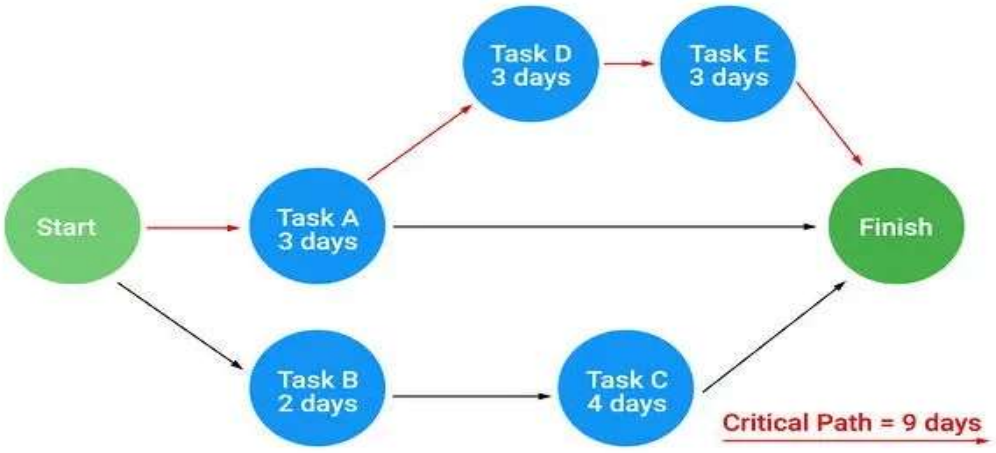


Fig 6: CPM chart

Critical Path Analysis components

Engineers use CPA to create a project model that includes the following features:

a. Activities are grouped and categorized in the Work Breakdown Structure (WBS)

b. Duration of each of the activities for example in given fig task A takes 3 days to complete.

c. **Float (slack):** time that defines how long you can delay the task before it affects the project schedule.

d. **Crash duration**: the minimum time required for scheduling the task.

e. The dependencies and connections between the activities

f. Deliverables and milestones

Using these components, the longest path can be estimated from allocated activities and milestones.

A. The earliest and latest start and finish dates:

The latest start date is the last moment when you can start the task.

The latest finish date is the last minute in which you can complete the task.

Advantages:

1. It uses deterministic time estimates
2. It figures out the activities which can run parallel to each other.
3. CPM provides demonstration of dependencies which helps in the scheduling of individual activities.
4. It shows the activities and their outcomes as a network diagram.
5. It helps in determining the slack time.
6. It is extensively used in industry.
7. It helps in optimization by determining the project duration.
8. It adjusts other activities (by allocating more or less resources)

Disadvantages of Critical Path Method (CPM):

1. The scheduling of personnel is not handled by the CPM.
2. In CPM, it is difficult to estimate the completion time of an activity.
3. The critical path is not always clear in CPM.
4. For bigger projects, CPM networks can be complicated too.
5. It also does not handle the scheduling of the resource allocation.
6. In CPM, critical path needs to be calculated precisely.

Working Principle:

It keeps the record of interdependencies of various CPM activities like

- Construction
- Time and Cost

- Critical path & slack.

  For example, It installing the plumbing, is on the critical path. You can shorten that task by hiring another plumber or by having the scheduled plumber work overtime if you have the budget for it

3.4.4 Work Breakdown Structure

It breaks project tasks into successively finer levels of Program such as - Project - Task – Work.

a. Firstly, the project managers and top-level management identifies the main deliverables of the project. Then, these main deliverables are broken down into smaller higher-level tasks and this complete process is done recursively to produce much smaller independent tasks. Generally, the lowest level tasks are the simplest and most independent tasks and take less than two weeks' worth of work. The efficiency and success of the whole project majorly depend on the quality of the Work Breakdown Structure of the project and hence, it implies its importance.

b. A Work Breakdown Structure includes dividing a large and complex project into simpler, manageable and independent tasks.

   Step1: The root of this tree (structure) is labelled by the Project name itself.

   Step2: It follows a Top-Down approach.

   Step-3: Identify the sub-activities of the major activities.

   Step-4: Repeat till undividable, simple and independent activities are created.

   Step5: specify work unit defines the shortest time span, specific start & end point, budget table in terms of money, resources. It can be assigned as an individual responsibility and hence can be re-scheduled.

   Step6: Each activity has duration constraint and consumes resources.

c. Purpose of WBS:

Its purpose is to make the project

- Manageable
- Independent
- Integral
- Measurable

d. **Example: WBS of commercial building construction project**

This phase-based work breakdown structure example of a construction project below handles the complexity of the project through the following network:

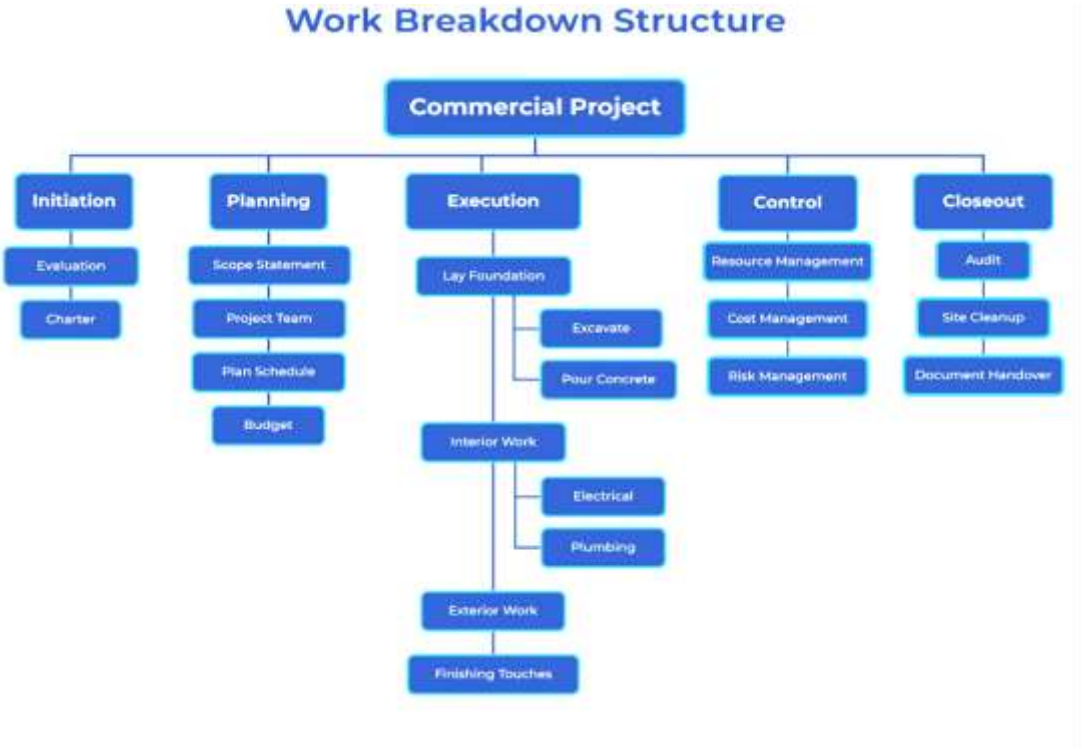


Fig 6: work Breakdown Structure

It shows the project phases, deliverables and work packages.

1. The top level is work breakdown structure which is final deliverable (in this instance, the construction project).
2. Second levels are project phases.
3. The third and lowest level shows work packages.

Advantages:

- It provide precise cost estimation of each activity.
- It provide estimated time that each activity will take.
- It provide easy management of the project.
- It provide proper organisation of the project by the top management.

Disadvantages:

1. Developing the work breakdown structure is not an easy task as it is time consuming and requires a lots of effort in case of complex projects.
2. In the development of the sequence, all the involved members need to participate and have their approval. So, it is not easy to implement.

3.5 Software Project Management Plan (SPMP)

Software Project Management plan involves various inputs, outputs and machinery to "do the right things, with the right tools, and in the right way". The various processes of SPMP are realized through various stages as shown in figure 2.

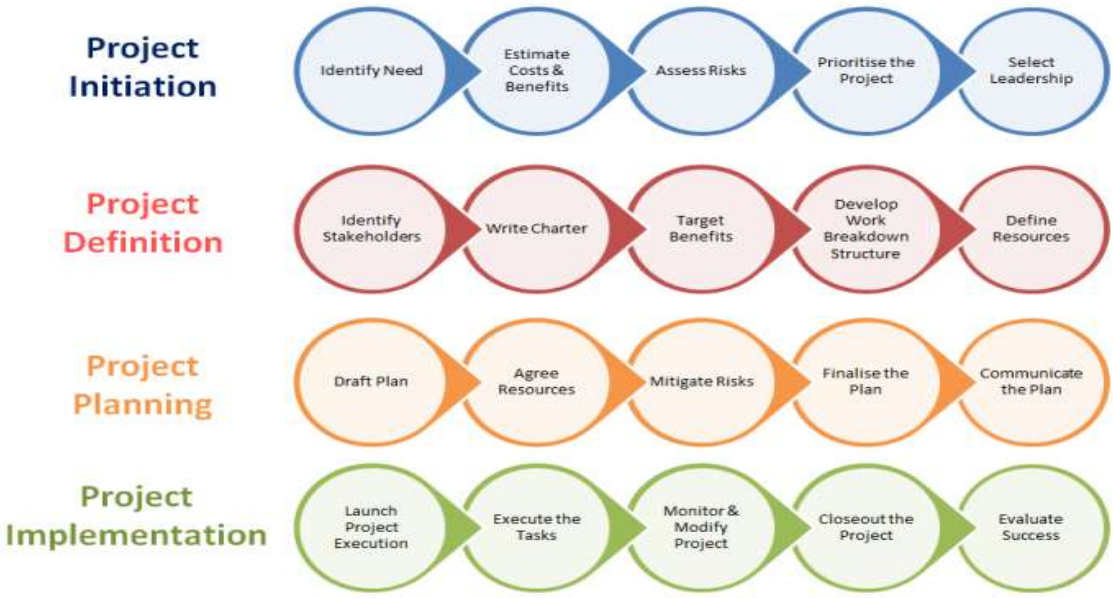


Figure 2 : Project management Processes

These processes are realized through the various SPMP stages.

Stage 1: Project Goals.

1. In this every team have an understanding of what must be accomplished.
2. It ends with a document that lists the goals with a short statement providing some detail.
3. It provides a description of the vital few requirements that define the goal as accomplished.
4. It prevents teams from performing unnecessary iteration and improvement on a goal that has been sufficiently accomplished.

Stage 2: **Define Project tasks and Activities**

5. Each goal or sub-set of goals is matched to the tasks required.
6. This is best done by listing the goals on the left side of a sheet of paper, then writing the tasks to their right.
7. The group should agree that the specify task will accomplish the goals as per required in the definitions for success.

Stage 3: Determine and verify resource requirements

1. Resources are those things which are needed to accomplish the project goals. Resources include, but are not limited to:

   • People • Time • Money • Space • Computers • Software • Others.

2. One of the most common mistakes project managers make is to underestimate the amount and type of resources required. This leads to projects that run over budget and fall behind schedule.
3. Determine what you need to get the job done correctly, on time, and on budget with "padding" of safety margin. For instance, one might multiply the estimated time to complete a task/project by say 20% to allow for additional time to deal with unexpected occurrences.
4. In some instances, it may be necessary to acquire resources as the project progresses, which is risky and should be avoided if possible.

Stage 4: Identify risks and develop mitigation (backup) plans

1. Projects always involve a finite amount of uncertainty and risks that may lead to problems during the project.
2. Risk management is important as it helps the team to complete the project while considering market risks.
3. Risk management reduces the likelihood and effects of risks.
4. For instance, an event causing project delays and deflect of responsibility, the resource re-allocation is done by the project manager.

Stage 5: Develop a schedule

1. A Gant chart, a schedule which plots the tasks, people responsible for these tasks, and a timeline, are useful as they allow the team to look at the architecture (structure) of the project and easily identify responsibilities.
2. The Gantt chart can serve as a visualization tool to see how tasks depend on each other.
3. The basic format of a Gantt chart consists of a listing of tasks on the left-hand side, followed by the start date, number of days to complete, and a finish date. Each task should be assigned one or more owners.
4. It is a graphical representation of the task duration in the context of the project time line. For instance: A sample Gant Chart can be made to look for conflicts of resources.

Stage 6: Execute the schedule

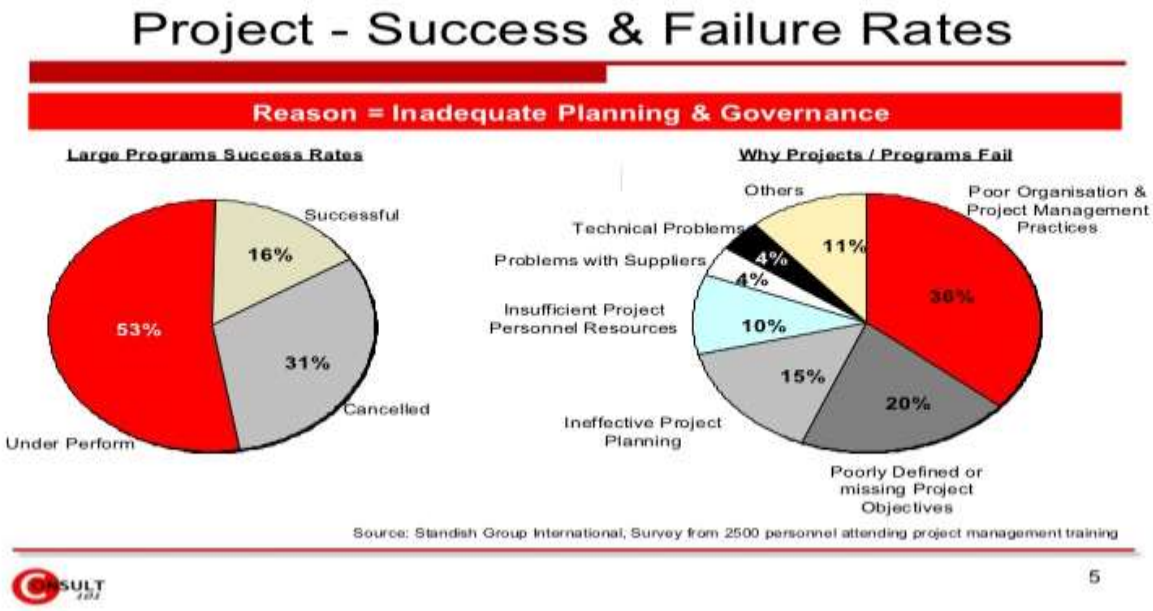
1. During this stage, the project manager is responsible for coordinating and temporarily shifting the resources to keep the team member on schedule.
2. Each group member should document their activities. Documentation is the responsibility of the team members and this can be facilitated by regular meetings (daily, weekly, monthly).

3. At the meetings the team should review the schedule and the status (complete or not complete) of the project goals.

Stage 7: Assessing Project Performance

1. It is good practice to evaluate the performance of the project team, where a learning and experience is gained.
2. It will help prevent similar problems in future projects.

Output of Framing SPMP:



3.6 Summary

Any project with the proper management has very little chances of failure. Various management activities of time, money and resource scheduling is carried out to deliver the project in time.

3.7 Suggested Readings

1. "Software Engineering: A Practitioner's Approach" 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.
2. "Software Engineering" by Ian Sommerville, Addison-Wesley, 7th Edition.
3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

3.8 Model Questions

1. Explain the various activities of Software Project management Plan in detail.
2. Explain various Project Scheduling Techniques with examples.
3. Compare any two Scheduling Techniques I detail.

Lesson-4

SOFTWARE PROJECT ESTIMATION AND RISK MANAGEMENT

Structure

4.0 Objectives

Software Project Estimation is the process of measuring and quantifying the complexity of the project in terms of various resources of effort, time and cost. To deliver the project in time various Problem and Process based estimation techniques are implemented through various product and process metrics.

4.1 Introduction

The product metrics are used to measure the internal attributes like usability, reliability, maintainability and portability of the software. The process metrics quantify the various activities of software development process and its environment.

4.2 Decomposition Techniques:

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data is incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

- Past Data/Past Experience

- Available Documents/Knowledge
- Assumptions
- Identified Risks
- **Software project estimation is a form of problem-solving, and in most cases, the problem to be solved is too** complex. For this reason, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems.
- As the effort (Time and Space Complexity) grows exponentially the Software Engineering amends this growth complexity to linear increase by using decomposition and abstraction tools.
- The decomposition approach was discussed from two different points of view: **decomposition of the problem and decomposition of the process.** The estimation uses one or both forms of partitioning. But before an estimate can be made, the project planner must understand the scope of the software to be built and generate an estimate of its "size."
- The accuracy of a software project estimate is predicated on a number of things: (1)the degree to which the planner has properly estimated the size of the product to be built; (2) the ability to translate the size estimate into the human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects); (3) the degree to which the project plan reflects the abilities of the software team; and (4) the stability of product requirements and the environment that supports the software engineering effort.

The four basic steps in Software Project Estimation are −

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

4.2.1 Problem Based Estimation: Definition:

a) It measures, quantifies, and predicts the complexity of the project through the triple constraints of effort, cost, and size for the validation of the final product.

b) To size the elements of software these factors are estimated using various baseline recognized metrics of LOC and FP.

The predicted accuracy of project estimate=Product size+ Translation of size into human effort+ Calendar Time and Rupees+ Stability of requirement and environmental factors.

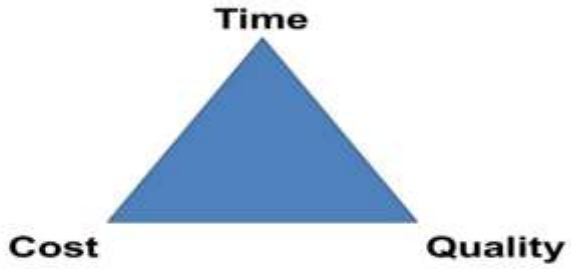


Fig 3: **Problem Based Estimation**

Problem Based Estimation is of two types:

A. Product Metrics

B. Process Metrics

Importance:

a) Uses Product Metrics to quantity the internal attributes like usability, reliability, maintainability and portability.

b) Uses Process metrics to quantify the software development process and its environment.

c) Handles the software Risks which can have adverse effects on cost, schedule and technical success of a project.

Several Techniques exist to assist software managers in making project size estimates like LOC and FP.

1. **Lines of Code:**

a) The simplest technique that measures the program length in KDSI (Kilo delivered source instructions).

b) It counts the number of source instructions where headers lines, comments, and blank lines are ignored.

c) One code line is one instruction.

Flaws:

i) Accurate estimation of Source lines is difficult at the beginning of the project, so expertise is required.

ii) Size can vary due to individual coding styles like using “switch” or “ladder -if”. So, language tokens like for, if, while keywords can be counted using Token Metric)

iii) A bad Software design may lead to excessive code lines.

iv) It does not estimate the functionality and complexity of system.

v) Two modules with different logic may have same LOC as it is language-dependent.

vi) Does not measure quality and efficiency. More number of lines does not mean code is efficient.

Example: To implement complex logic and to introduce high quality, sometimes lengthy code is written like calculating Factorial with and without Recursion.

Function Points:

A) Proposed by Albrecht [1973], it measures the software size by counting the number of different features/ Functions carried out by project

1. A program supporting more functions will be better than a program supporting fewer features.
2. Size can be estimated before actual implementation as the developer knows the different functions supported by project at early stages of project development.
3. Effort is saved as detailed design is available before actual implementation.
4. Used as basic data structure where productivity metrics are compiled to quantity the size.

Implementation of FP:

1. Identification and counting of unique function types.
2. Assigning functions, a weight to quantify their complexity from 3 to 15.
3. Albrecht derived weights empirically and validated through following parameters.

Type	Simple	Average	Complex
Input (External files)	3	4	7
Output (Reports and Messages)	3	5	6
Enquiry (Queries)	2	4	8
Interfaces	6	9	11
Internal files (Hidden files)	8	11	15

Table-1: Parameter Grading

4. Computing FP= multiplying each function count * weighting factor.

   UFP (Unadjusted FP) = weighed sum of functions/ Project characteristics.

   =∑ (Number of Inputs) *4+(Outputs*5) +(Inquiries *4) +(Interfaces *10) + (Internal files *10)

a) UFP is calculated to accommodate the actual contribution of various complexity parameters. Hence the complexities are graded (Table-1)

5. (Technical Complexity Factor) TCF is calculated to check the contribution of Technical Characteristics like response time, Transaction Rates which can influence the development effort.
6. Albrecht identified 14 parameters and weighted them from 0 – No influence for their influenceto 6: Strong influence.

Hence TCF=0.65+0.01+DI (Degree of Influence)

DI: 0 to 84= weighted sum of (Tech. Parameters * influence Value) =(14 Parameters *6=84)

So TCF is 0.65 to 1.49(0.65+0.01*84) = (0.65+0.84) =1.49

Flaws

1. Does not incorporate algorithmic complexity.
2. Sometimes a TCF is not the correct estimation of 14 parameters due to noise, interrupted communication, different hardware complexity. Like online and offline data entry and updation, data communication Time.

Deriving Productivity Metrics:

Productivity: FP/ Person -Month

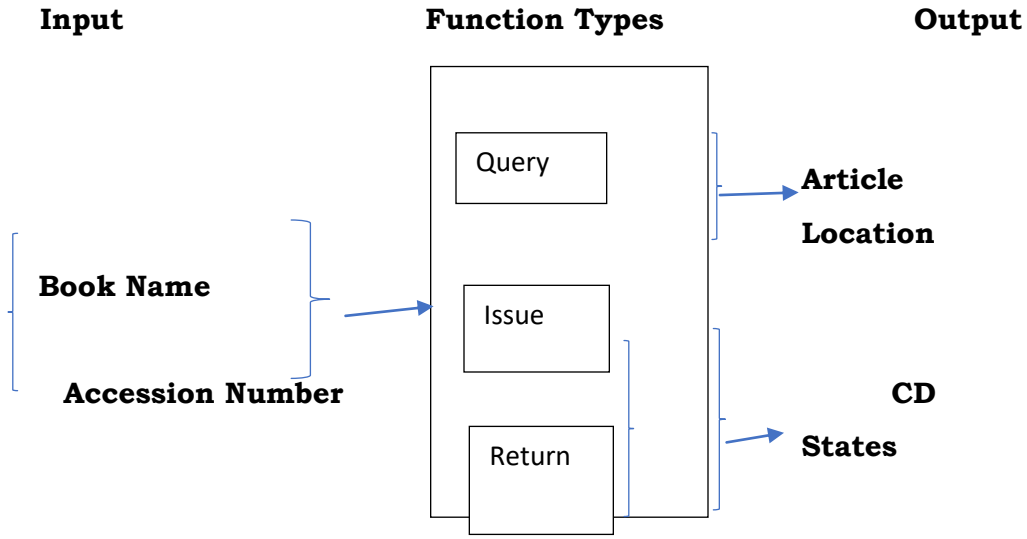
Quality =No. of defects/ FP

Documentation= No. of Document Pages/FP

Benefits: 1. Language Independent send restricted to code.

2. Good measure as problem- specific parameters are considered.
3. Used to judge productivity, Quality and Documentation.

Ex: **Library Automation System:**



4.2.2 Process-Based Estimation: It is the process of finding an estimate, or approximation, which is a value that is usable for some purpose even if input data may be incomplete, uncertain, or unstable. The value is not usable because it is derived from the best information available.

A. Constructive Cost Model (COCOMO):

Constructive Cost Model (**COCOMO**) is a procedural **software** cost estimation model developed by Barry W. Boehm. The COCOMO cost estimation model is used by thousands of software project managers and is based on a study of hundreds of software projects. Unlike other cost estimation models, COCOMO is an open model with features of:

- The underlying cost estimation equations
- Every assumption made in the model (e.g., "the project will enjoy good management")
- Every definition (e.g., the precise definition of the Product Design phase of a project)
- The costs included in an estimate are explicitly stated like project managers are included, secretaries are not.

Because COCOMO is well defined, and because it doesn't rely upon proprietary estimation algorithms, it offers these advantages to its users:

- COCOMO estimates are more objective and repeatable than estimates made by methods relying on proprietary models
- COCOMO reflects your software development environment, and to produce more accurate estimates

COCOMO Model: Implementation

The most fundamental calculation in the COCOMO model is the use of the Effort Equation to estimate the number of Person-Months required to develop a project. Most of the other COCOMO results, including the estimates for Requirements and Maintenance, are derived from this quantity. The implementation steps are:

a. Source Lines of Code

The COCOMO calculations are based on your estimates of a project's size in Source Lines of Code (SLOC). SLOC is defined such that:

- Only Source lines that are DELIVERED as part of the product are included -- test drivers and other support software is excluded
- SOURCE lines are created by the project staff -- code created by applications generators is excluded
- One SLOC is one logical line of code
- Declarations are counted as SLOC
- Comments are not counted as SLOC

The original **COCOMO 81 model** was defined in terms of Delivered Source Instructions (DSI), which are very similar to SLOC. The major difference between DSI and SLOC is that a single

Source Line of Code may be several physical lines. For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI.

b. The Scale Drivers

In the **COCOMO II model**, some of the most important factors contributing to a project's duration and cost are the Scale Drivers. You set each Scale Driver to describe your project; these Scale Drivers determine the exponent used in the effort equation.

The 5 Scale Drivers are:

1 Precedence

2 Development Flexibility

3 Architecture / Risk Resolution

4 Team Cohesion

5 Process Maturity

c. Cost Drivers

- COCOMO II has 17 cost drivers to assess the project and development environment. The cost drivers are multiplicative factors that determine the effort required to complete your software project. For example, if your project is to develop a software that controls an airplane's flight, you would set the Required Software Reliability (RELY) cost driver to Very High. That rating corresponds to an effort multiplier of 1.26, meaning that your project will require 26% more effort than a typical software project.

COCOMO II Effort Equation

- The COCOMO II model estimates the required effort (in Person-Months PM) based primarily which is on the estimate of the software project's size (as measured in thousands of SLOC, KSLOC)): Hence:

  $$\text{Effort} = 2.94 * \text{EAF} * (\text{KSLOC})^{E}$$

  Where

  EAF: Effort Adjustment Factor derived from the Cost Drivers

  E : exponent derived from the five Scale Drivers

  As an example, a project with all Nominal Cost Drivers and Scale Drivers would have an EAF of 1.00 and exponent, E, of 1.0997. Assuming that the project is projected to consist of 8,000 source lines of code, COCOMO II estimates that 28.9 Person-Months of effort is required to complete it:

  $$\text{Effort} = 2.94 * (1.0) * (8)^{1.0997} = 28.9 \text{ Person-Months}$$

4.3 Risk Management

The main purpose of risk management is to identify and manage the risks associated with a software project and solve the problem. Estimating the risks that can affect the project schedule or the quality of the software being developed and taking action to avoid the risk is the important task of a project manager. Identifying and preparing plans to reduce their impact on the project is called risk management.

4.3.1Risk Management Activities:

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

Risk Assessment

The objective of risk assessment is to detect the risks in the condition of project loss.. For risk assessment, first, every risk should be rated in two methods:

a. The possibility of a risk coming true (denoted as r).

b. The consequence of the issues relating 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.

4.3.2. 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.

To identify the significant risk, that affect a project. It is necessary to categorize into the different risks of classes.

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

Technology risks: Risks that assume from the software or hardware technologies which are used to develop the system.

People risks: Risks that are connected with the person in the development team.

Organizational risks: Risks that assume from the organizational environment where the software is being developed.

Tools risks: Risks that assume from the software tools and other support software used to create the system.

Requirement risks: Risks that assume from the changes to the customer requirement and the process of managing the requirements change.

Estimation risks: Risks that assume from the management estimates of the resources required to build the system

4.3.3. Risk Analysis:

During the risk analysis process, 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, numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

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%).

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.

Risk Management Planning:

There are three main methods to plan for risk management:

a. **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.

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

c. **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.

d. **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)

4.3.4 Risk Classification: Software Risks:

The key purpose of classifying risk is to get a collective viewpoint on a group of factors. These are the types of factors that will help project managers to identify the group that contributes the maximum risk. The best and most scientific way of approaching risks is to classify them based on risk attributes. Risk classification is considered as an economical way of analyzing risks and their causes by grouping similar risks together into classes.

Software risks could be classified **as internal or external.** Those risks that come from risk factors within the organization are called internal risks whereas the external risks come from out of the organization and are difficult to control.

Internal risks are project risks, process risks, and product risks.

External risks are generally business with the vendor, technical risks, customers' satisfaction, political stability and so on. In general, there are many risks in the software engineering which is very difficult or impossible to identify all of them. Some of most important risks in software engineering project are categorized as software requirement risks, software cost risks, software scheduling risk, software quality risks, and software business risks. These risks are explained as below:

A. SOFTWARE REQUIREMENT RISKS

1. Lack of analysis for change of requirements.
2. Lack of report for requirements
3. Inadequate of requirements
4. Invalid requirements

B. SOFTWARE COST RISKS

1. Lack of good estimation in projects
2. Human errors
3. Lack of monitoring and testing
4. Complexity of architecture
5. Personnel change, Management change, technology change, and environment change

SOFTWARE SCHEDULING RISKS

1. Inadequate budget
2. Inadequate knowledge about tools and techniques
3. Lack of enough skill
4. Lack of good estimation in projects

C. SOFTWARE QUALITY RISKS

1. Inadequate documentation
2. Lack of project standard
3. Inadequate budget
4. Inadequate knowledge about techniques, programming language, tools

4.4 Strategies for Risk Management

During the software development process, various strategies for risk management could be identified and defined according to the amount of risk influence. Based upon the amount of risk influence in software development project, risk strategies could be divided into three classes namely careful, typical, and flexible.

a. Careful risk management strategy is projected for new and inexperienced organizations whose software development projects are connected with new and unproven technology.

b. Typical risk management strategy is well-defined as a support for mature organizations with experience in software development projects and used technologies, but whose projects carry a decent number of risks.

c. flexible risk management strategy is involved in experienced software development organizations whose software development projects are officially defined and based on proven technologies.

4.5 Summary

In this way, software risk management, risks classification, and strategies for risk management are clearly described in this chapter. If risk management process is in place for each and every software development process then future problem could be minimized or completely eradicated. Hence, understanding various factors under risk management process and focusing on risk management strategies explained above could help in building risk free products in future.

4.6 Suggested Readings

1. “Software Engineering: A Practitioner’s Approach” 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.
2. “Software Engineering” by Ian Sommerville, Addison-Wesley, 7th Edition.
3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

4.7 Model Questions

1. Draw difference between Problem and Process based estimation.
2. Explain COCOMO cost estimation model with example.
3. Explain various risk management activities in detail.

Lesson-5

SOFTWARE DESIGN

Structure

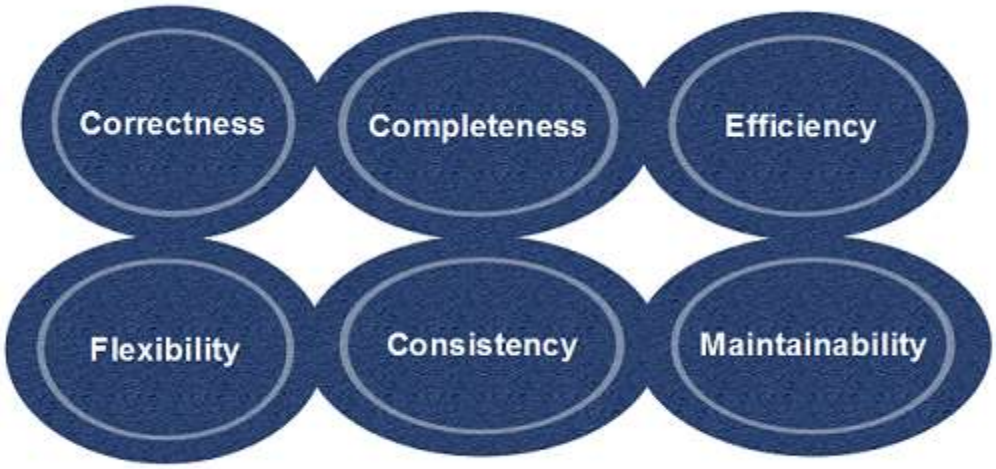
Software Design:

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, that is easily implementable using programming language.

The software design phase is the second step which is in SDLC (Software Design Life Cycle), which transfers the concentration from the problem domain to the solution domain. In software design, we consider the system to be a set of components or modules having clearly defined behaviours and boundaries.

5.0 Objectives of Software Design:

The design is a representation of a product or a system with sufficient details for implementation**.**



Objectives of Software Design

Various Objectives of software designs are:

1. **Correctness:** Software design should be validated and verified as per requirements.
2. **Completeness:** The design should have all components like data structures, modules, and external interfaces to depict its complete behaviour.
3. **Efficiency:** Resources should be used efficiently by the program.
4. **Flexibility:** It should be modifiable to changing needs.
5. **Consistency:** There should not be any inconsistency in the design for uniqueness.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers also.

1. Software Design Process:

Software Design Process is a high-rank, technology-independent concept that describes a system that will be able to accomplish the established tasks in the requirement analysis phase.

According to the IEEE, Design is defined as the "process of defining the architecture, components, interfaces and other characteristics of a system along with their results."

5.1 Importance of Software Design Process

a. Software Design gives a foundation to construct your software structure. Hence, it is an essential step.

b. It precedes creating or enforcing the product in every engineering field.

   For example – when constructing a building, it is unbelievable and impractical for the builders to go straight to the ground and start the construction before establishing detailed designs with engineers, as it would be risky.

c. Design is initial and most crucial phase of the software development methodology.

d. Design facilitates to settle between suggested solutions and available resources.

5.2 Principles of Good Software Design Process

Many principles are employed to organize, coordinate, classify, and set up software design process and its structural components. Software Designs become some of the most convenient designs when the following principles are applied. They help to generate remarkable User Experiences and customer loyalty.

The principles of a good software design are:

1. Modularity
2. Coupling
3. Abstraction
4. Anticipation of change
5. Simplicity
6. Sufficiency and completeness

1. **Modularity**
   Dividing a large software project into smaller functional portions/modules is known as modularity. It is the key to scalable and maintainable software design. The project is divided into various components and work on one component is done at once. It becomes easy to test each component due to modularity. It also makes integrating new features with more accessibility.

2. **Coupling**
   Coupling refers to the extent of interdependence among software modules and how closely two modules are connected. Low coupling is a feature of good design. With low coupling, changes can be made in each module individually, without changing the other modules.

3. **Abstraction**
   The process of identifying the essential behaviour by separating it from its implementation and removing irrelevant details is known as Abstraction. The inability to separate essential behaviour from its implementation will lead to unnecessary coupling.

4. **Anticipation of Change**

   The demands of software keep on changing, resulting in continuous changes in requirements as well. Building a good software design consists of its ability to accommodate and adjust to changes comfortably.

5. **Simplicity**
   The aim of good software design is simplicity. Each task has its own module, which can be utilized and modified independently. It makes the code easy to use and minimizes the number of setbacks.

6. **Sufficiency and Completeness**

   A good software design ensures the sufficiency and completeness of the software concerning the established requirements. It makes sure that the software has been adequately and wholly built.

5.4 Stages of Software Design Process

The stages of software design process are:

Stage 1: Understanding of project requirements

Stage 2: Research and Analysis

1. Interviews
2. Focus groups
3. Survey

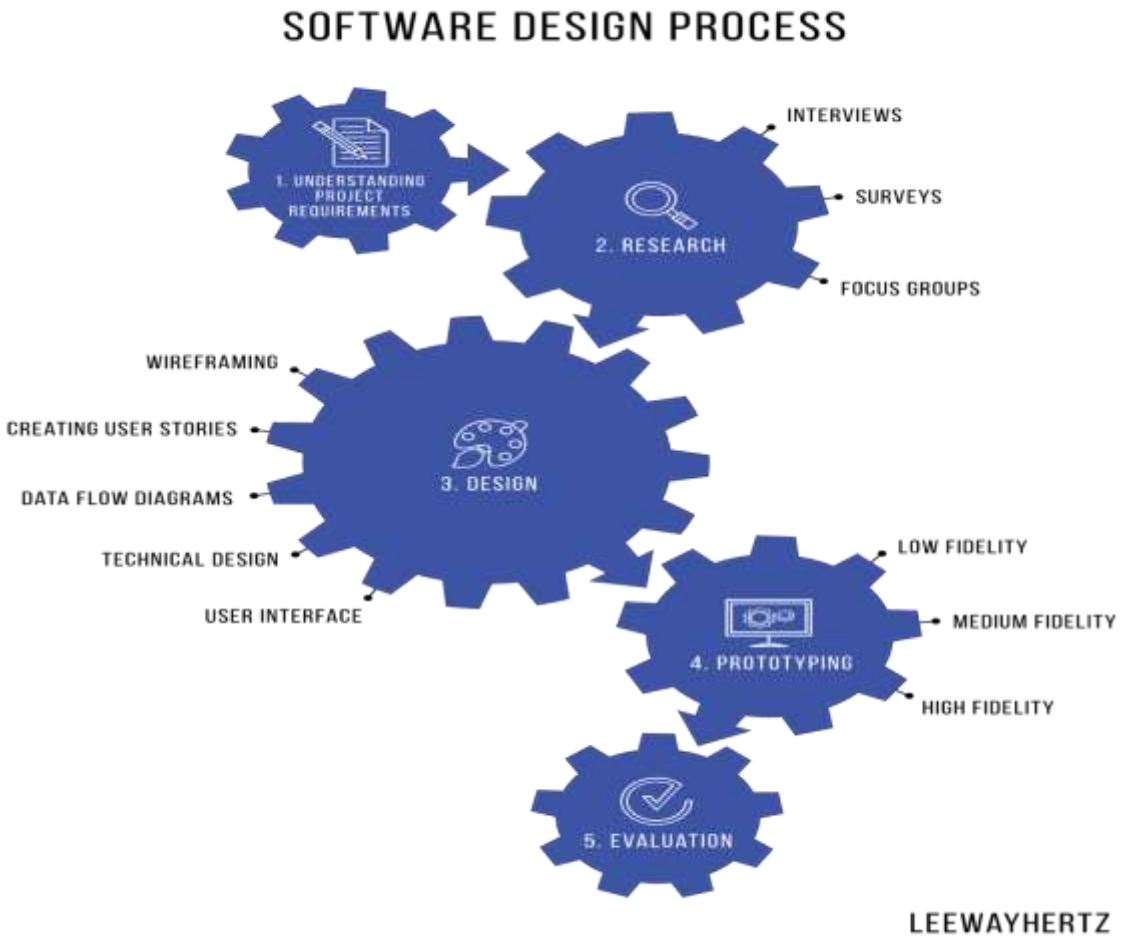
Stage 3: Design

1. Wireframing
2. Creating user stories
3. Data flow diagrams
4. Technical Design
5. User Interface

Stage 4: Prototyping

1. Low Fidelity Prototyping
2. Medium Fidelity Prototyping
3. High Fidelity Prototyping

Stage 5: Evaluation



STAGE 1: UNDERSTANDING PROJECT REQUIREMENTS

Before starting designing software, there is a need to make sure that all the project requirements must know. These requirements can comprise users' difficulties, and what issues need to address while creating a software design specification.

STAGE 2: RESEARCH AND ANALYSIS

This stage comprises research about the target audience to create, analyse the data collected, and determine the user design's basic requirements. The decisions made in this stage become the foundation of building the entire software.

Thorough research can be conducted through: -

Interviews – Sitting with users and questioning them about their problems can provide first-hand information and honest opinions about the various issues.

Focus Groups –To conduct and observe discussions about a specific issue that prevails with software design. It helps to understand their preferences, opinions, and their difficulties more clearly.

Surveys –

a. By sending out questionnaires to the target audience about a specific issue, it is easy to find their opinions and attitudes towards their problems and preferences in software design.

b. It is less time-consuming than interviews and focus groups.

c. Through your research, you can analyse it and determine the factors that focus on software design to ease the target audience's needs.

STAGE 3: DESIGN:

After gathering and analysing data, you start to design your software. This stage majorly consists of the following components: -

1. Wireframing:

Wireframing is a vital step in the designing process. It helps to gain a deep understanding of project structure before the creative influences take place. This step helps to save time and effort in the later stages of the process.

These are important:

1. Wireframes act like the "skeleton" of software design to determine the basic structure, elements, features, navigation, and look of the software before adding content and graphics.
2. They show each page's bare essentials – what goes where, how the page looks and may act as a prototype of the actual software design.
3. Wireframes are easy to edit. The wireframes can be edited until the structure matches their business needs.

2. Creating User Stories:

1. User stories are a raw, casual, and natural description of the features of a software.
2. All user stories are a component of the agile approach.
3. User stories should be written from the user's perspective, as they convey how a user is going to employ the software.
4. User stories should be discussed and collaboratively worked on so that all details are taken into consideration.
5. User stories are refined over the time.

For Example:

Suppose you are developing a food delivery app. The users would be both the delivery person as well as the customers. Some of the user stories may look like the following: -

i) As a customer, I want to link my debit card to my profile for quicker payments.

ii) As a delivery person, I want to upload my preferences for delivery areas.

iii) As a customer, I want to give feedback on the quality of food received.

3. **Data Flow Diagrams (DFD)**

DFD are the conceptual explanation of system details. They are of 3 types: -

A. **0-level DFD: –** It is also known as the fundamental system model. It shows the complete system requirement in a single bubble, with inward and outward arrows denoting data movement.

B. **1-level DFD: –** Multiple processes/bubbles are shown at this level. It shows the system's main aims here and breaks down the 0-level DFD into more detail.

C. **2-level DFD: –** It takes 1-level DFD in more depth. It records specific details about the software's functioning of each of the different functional.

D. **Technical Design**

Technical Design explains how to convert the functionality of a system into programming code.

a. It is the backbone of the project as it guides all the implementation.

b. Technical Design document lists down every technical detail of the entire software design.

c. It is also referred to as "technical blueprint/as it includes communication interfaces, input and output details, software requirements, software architecture.

d. It provides a detailed description of hardware components, data structure and data flow **User Interface:**

   a. The user interface comprises of all the parts of software with which a user can interact.

   b. It aims at making the use of the user interface enjoyable and pleasurable.

   c. There are three types of user interfaces: -

1. **Graphical User Interfaces (GUI) –** The users interact with graphics on digitally regulated panels. e.g., Computer desktop.
2. **Voice Controlled Interfaces (VUI) –** The users interact through their voices with these interfaces. e.g., Apple's Siri, Amazon's Alexa
3. **Gesture-Based Interfaces (GBI) –** Users engage with these interfaces through bodily movements. e.g., VR Games

STAGE 4: PROTOTYPING

A prototype is the draft of a product that gives a good visual representation of the final product. A prototype is more detailed than a wireframe.

a. Prototypes help to test layouts and figure out if everything matches with the pre-established requirements and gather user feedback.

b. They also assist in saving time and money.

c. It helps in brainstorming for new ideas and trying out new ideas before settling on the final design.

d. Wireframes and task flows can be used with various computer-based tools to present a basic but more formal prototype to the client.

e. Prototypes are required when there is a need for superior visual and functional accuracy.

STAGE 5: EVALUATION:

This stage of the software designing process involves user testing.

a. It evaluates the degree of requirement fulfilment and checks repeatedly if the software design is smooth, simple, and direct.

b. This stage is essential to figure out the design problems and debug them before launching the design

Example tools used for Software Design:

Many tools can be used for designing software. The six most effective and commonly used tools are: -

1. Draw.io
2. Jira
3. Mockflow
4. Sketch
5. Marvel
6. Zeplin

- **Draw.io**
  Draw.io is a tool used to create diagrams in Confluence and Jira. You can easily create flowcharts, process diagrams, ER diagrams and much more. It also assists you in designing your own unique shape libraries.
- **Jira**
  Jira is a highly efficient software development tool used to plan, track, release and report your work. It also allows you to choose a workflow or make your own workflow.
- **Mockflow**
  Mockflow is a web-based wireframe tool for designers to plan, build and share their work. It is cloud-based and provides its users with an extensive library of mock components, icons, stickers, etc.
- **Sketch**
  The sketch is a vector graphic editor for MAC users. It is used for the user interface, mobile, web and icon design.

- **Marvel**
  Marvel provides intuitive designing and prototyping tools that help software designers to wireframe, design and prototype fast.
- **Zeplin**
  It provides a platform for designers to share, organize and collaborate on their designs. It facilitates better collaboration between designers and engineers.

5.5 Design Failures

1. Design rules are the guidelines rather than methods in the mathematical sense.
2. Different designers create quite different system designs hence affects efficiency.
3. Ambiguously stated design leads to misinterpretation at the implementation stage
4. In case of failure to consider potential design changes for extension and contraction leads to overly constrained and inefficient designs
5. They do not help much with the early, creative phase of design. Rather, they help the designer to structure and document his or her design ideas.
6. Sometimes undocumented design decisions may result in a non-integral and inconsistent system

5.6 Design Remedies:

Various techniques are employed to generate reliable and consistent design.

5.6.1 Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out the task(s) independently.

1. Designers should design modules such that they can be executed and compiled separately and independently.
2. Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy.

5.6.2 Concurrency

Concurrency provides the capability of the software to execute more than one part of code in parallel to each other.

1. concurrency is implemented by splitting the software into multiple independent units of execution, like modules, and executing them in parallel.
2. Its efficiency depends on programmers and designers capability to recognize those modules, which can be made for parallel execution.

Example

The spell check feature in the word processor is a module of the software, which runs alongside the word processor itself.

5.6.3 Coupling and Cohesion

When a software program is modularized, there are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion: Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

Coupling: Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program is.

5.6.4 Design Verification

a. The output of the software design process is design documentation, pseudo-codes, detailed logic diagrams, process diagrams, and a detailed description of all functional or non-functional requirements.

b. There outputs are verified through associated tools and are validated through design review.

c. By structured verification approach, reviewers can detect defects to improve accuracy and quality.

5.7 Summary

Software design is a crucial activity in the development of any software as its the final inner and outer architecture based on which the project will be coded, tested and implemented for the final delivery. It has to work on the final provided time and budget resources for in time delivery.

5.8 Suggested Readings

1. “Software Engineering: A Practitioner’s Approach” 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.
2. “Software Engineering” by Ian Sommerville, Addison-Wesley, 7th Edition.
3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

5.9 Model Questions

Q1. Explain the design process in detail.

Q2. What is the purpose of structured analysis and design?

Q3. Which are the structured analysis and design tools? Explain them briefly.

Lesson-6

STRUCTURED ANALYSIS AND DESIGN TOOLS

Structure

6.0 Objectives

Many design methods support comparable views of a system tools

- Structured methods are sets of notations for expressing a software design and guidelines for creating a design
- Can be applied successfully because they support standard notations and ensure design follows a standard form

6.1 Introduction

- Structured methods may be supported with CASE tools
- Well-known methods include Structured Design (Yourdon), JSD (Jackson Method), DFD(data flow diagrams), DD (Data dictionary) and ERD(Entity Relationship Diagrams)
- A structural view showing system components
- and their interactions
- A data flow view (data flow diagrams) shows the data transformations
- An entity-relation view describes the logical data structures

Definition: Structured Analysis and Structured Design (SASD) is a diagrammatic notation that is designed to help people understand the system. The basic goal of SASD is to improve quality and reduce the risk of system failure.

Objectives of SAAD:

1. It establishes concrete management specifications and documentation. It focuses on the strength, flexibility, and maintainability of the system.
2. Many design methods support comparable views of system tools. These are sets of notations for expressing a software design and guidelines for creating a design
3. Structured methods may be supported with CASE tools
4. Well-known methods include Structured Design (Yourdon), JSD (Jackson Method), DFD (data flow diagrams), DD (Data dictionary), and ERD (Entity Relationship Diagrams)
5. This structural view shows system components and their interactions.
6. A data flow view (data flow diagrams) shows the data transformations.
7. An entity-relation view describes the logical data structures
8. Basically, this approach of SASD is based on the Data Flow Diagram, that focuses on well-defined system boundary.

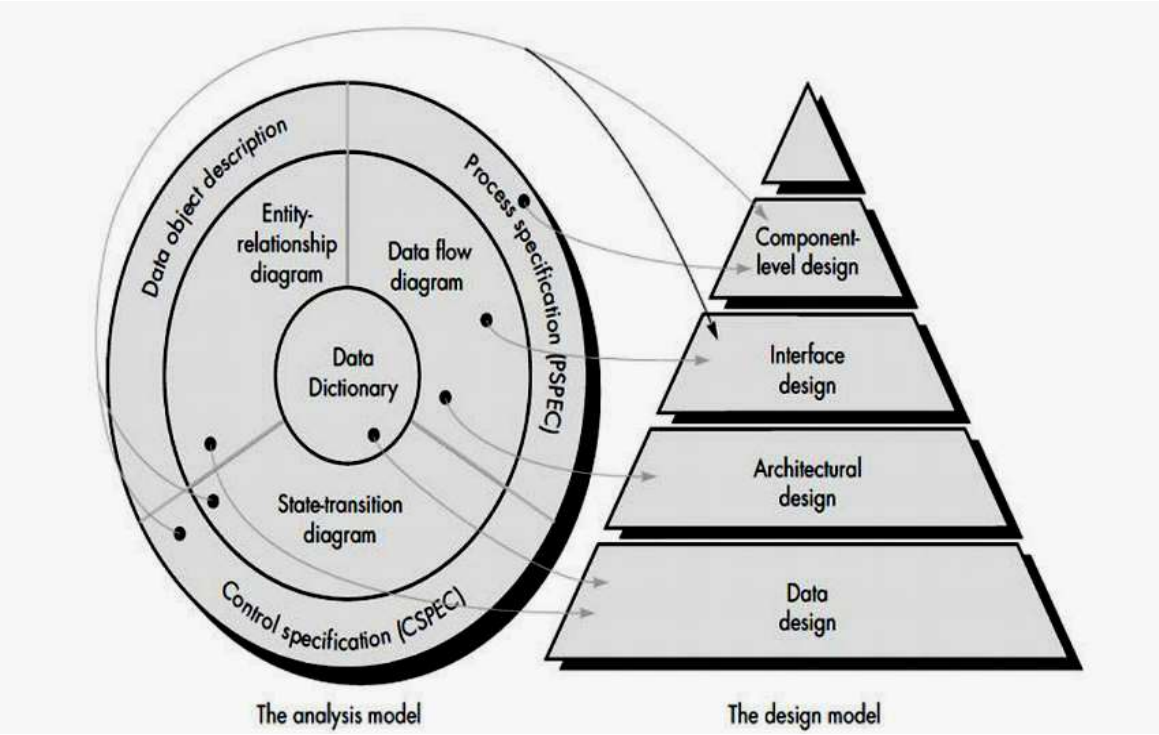


Fig1: Translating the analytical model into a software design

3. The various components of SASD technology are:

1. System
2. Process
3. Technology

It involves 2 phases as depicted in fig1:

1. **Analysis Phase:** It uses Data Flow Diagram, Data Dictionary, State Transition diagram, and ER diagram.
2. **Design Phase:** It uses Structure Chart and Pseudo Code.

Goals and Benefits of SASD:

The major goals of SASD are to improve the quality and reduce the risk of failure. The various goals are:

1. Need to obtain a clear and complete specification.
2. Documentation for the system.
3. Improves the quality and reduces the risk of system failure Establishes specifications and complete requirements.
4. It focuses on reliability, flexibility, reusability, robustness, maintainability of the system.

Benefits of SASD

1. SASD creates the map of the system, and these maps can be used for traceability, maintenance, or enhancement activities in the future.
2. It places emphasis on analysis and design activities, rather than implementation.
3. This encourages the project team to be thoughtful about the system's fundamental purpose and the engineering details.
4. The project team develops software documentation as they progress though development.

6.2 Data Flow Diagram

The data flow diagram is a graphical representation of the flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow, and stored data.

There is a prominent difference between DFD and Flowchart. The flowchart depicts the flow of control in program modules. DFDs depict the flow of data in the system at various levels. DFD does not contain any control or branch elements. Various features of DFD are:

1. The model describes how the data flows through the system.
2. It incorporates the Boolean operator and link data flow when more than one data flow may be input or output from a process.

For example: if we have to choose between two paths of a process, we can add a operator.

6.2.1 Types of DFD

Data Flow Diagrams are either logical or physical.

- **Logical DFD** - This type of DFD concentrates on the system process and flow of data within the system. For example, in a Banking software system, how data is moved between different entities of pension cell, cheque section and SBI facilities.

- **Physical DFD** - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and closer to the implementation.

6.2.2 DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components -

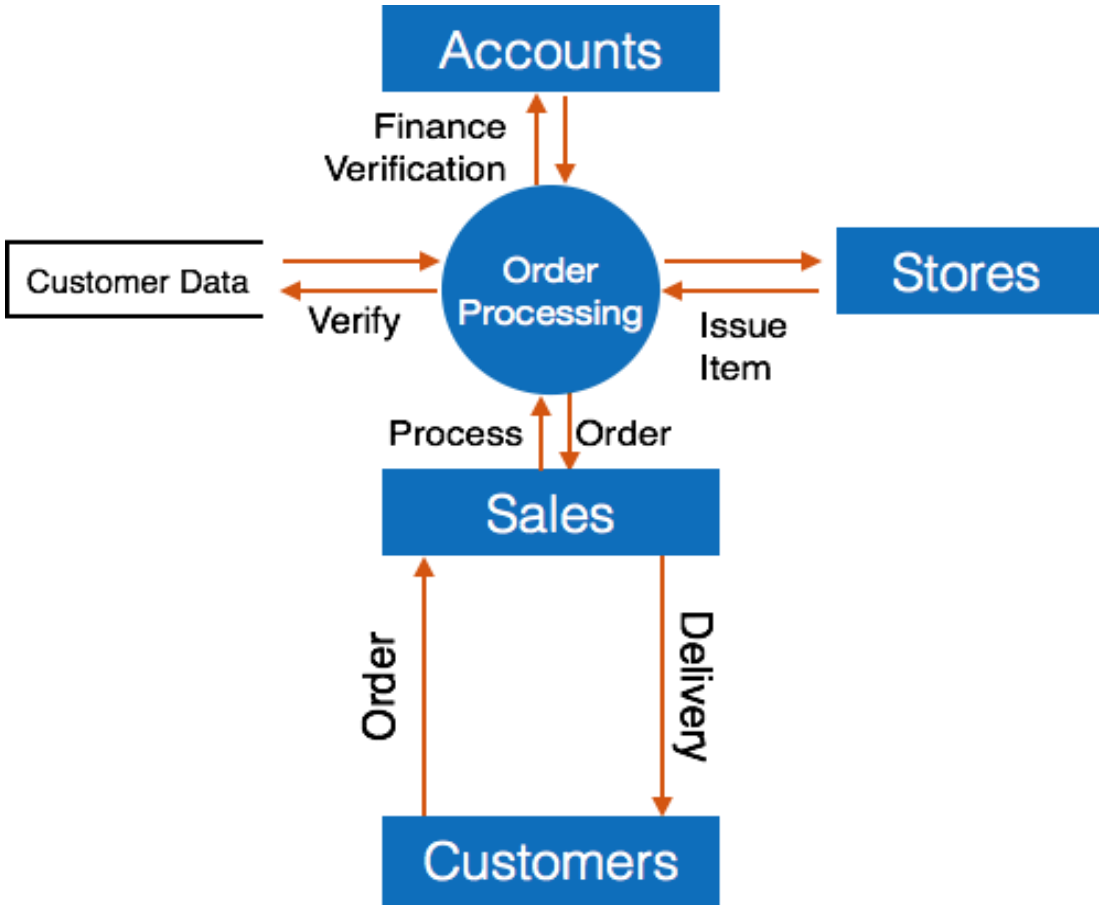
- **Entities** - Entities are the source and destination of information data. Entities are represented by rectangles with their respective names.
- **Process** – Activities, and actions taken on the data and are represented by circle or rounded edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with the absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

6.2.3 Levels of DFD: There are three levels of DFD

- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.
- **Example**:

- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.

- **Example:**



- **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level 1. Higher-level DFDs can be transformed into more specific lower-level DFDs with a deeper level of understanding unless the desired level

6.3 Data Dictionary:

The content that is not described in the DFD is described in the data dictionary. It defines the data store and relevant meaning. It also includes descriptions of data elements that flow external to the data stores.

Types of data dictionary:

1. **Physical DD**: shows the flows between process and entities.
2. **Logical DD:** A logical data dictionary contains system names, whether they are names of entities types, relations, attributes and services.

Features:

1. The data dictionary is an organized as listing of all data elements that are pertinent to the system

2. Defines rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores, and even intermediate calculations.
3. Today, the data dictionary is always implemented as part of a CASE "structured analysis and design tool."
4. Although the format of dictionaries varies from tool to tool, most contain the following information:

- **Name**—the primary name of the data or control item, the data store, or an external entity.
- **Alias—**other names used for the first entry.
- **Where-used/how-used**—a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity
- **Content description**—a notation for representing content.
- **Supplementary information**—other information about data types, pre-set values (if known), restrictions or limitations, and so on.

Example, a data dictionary entry may represent that the data gross pay consists of the entity components are.

gross Pay = regular Pay + overtime Pay

Data Dictionary Operators

The mathematical operators used within the data dictionary are defined in the table:

Notations	Meaning
x=a+b	x includes of data elements a and b.
x=[a/b]	x includes of either data elements a or b.
x=a x	includes of optimal data elements a.
x=y[a]	x includes of y or more occurrences of data element a
x=[a]z	x includes of z or fewer occurrences of data element a

Example of Data Dictionary:

The data dictionary entry begins as follows:

name: telephone number

aliases: none

where used/how used: asess against set-up (output) dial phone (input)
description:

1. telephone number = [local number| long distance number]
2. local number = prefix + access number
3. long distance number = 1 + area code + local number
4. area code = [800 | 888 | 561]
5. prefix = *a three-digit number that never starts with 0 or 1*
6. access number = * any four number string *

This content description is expanded until all composite data items have been represented as elementary items.

For example, the definition of area code indicates that only three area codes (two toll-free and one in city) are valid for this system.

Limitation of Data Dictionary:

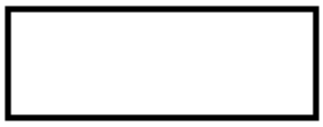
For large computer-based systems, the data dictionary grows rapidly in size and complexity. In fact, it is extremely difficult to maintain a dictionary manually. For this reason, CASE tools should be used.

6.4 Entity Relationship Diagrams

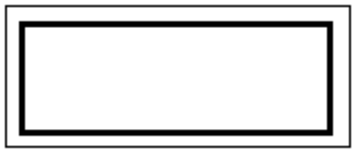
Definition:

An ERD is a logical representation of an organization's data used in database design. It basically describes the relationship between different entities of data store. There are three primary components and key features of ER are:

- **Entities – D**ata are represented by rectangles
- **Attributes –** Characteristics of entities are listed within entity rectangles
- **Relationships – R**elationships among entities are represented by lines
- An entity is an object or concept about which you want to store information.

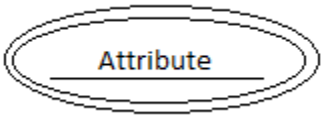


- A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



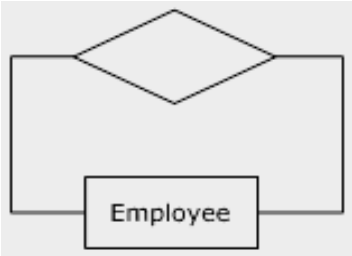
- A key attribute is the unique, distinguishing characteristic of the entity.

- A multivalued attribute can have more than one value.



- A derived attribute is based on another attribute.

- **Cardinality** specifies how many instances of an entity relate to one instance of another entity.
- **Recursive relationship:** In some cases, entities can be self-linked. For example, employees can supervise other employees.



For Example:

In this example Student is an entity having different attributes such as Name, DOB, Phone_Number, Age, Street, Roll_number.

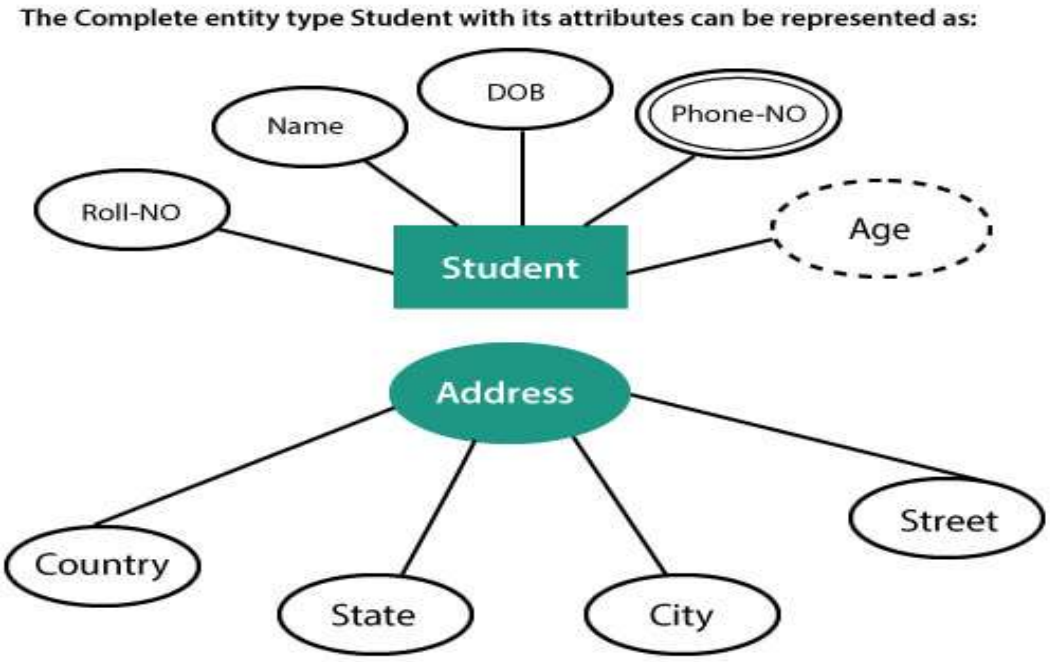


Fig: Complete Entity type.

Advantages and Disadvantages

Advantages:

- Straightforward relation representation
- Easy conversion for E-R to other data model
- Graphical representation for better understanding

Disadvantages:

- No industry standard for notation
- No representation of data manipulation
- Loss of information

6.5 Summary

The application of Structured Analysis and Design Tools is the effective way to get the inner details of project modalities in terms of advantages and disadvantages of various design methods. The various structured methods of DFD, DD and ER diagrams are implemented through various sets of notations to layout the guidelines for the creation of software design.

6.6 Suggested Readings

1. "Software Engineering: A Practitioner's Approach" 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.
2. "Software Engineering" by Ian Sommerville, Addison-Wesley, 7th Edition.

3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

6.7 Model Questions

1. What is the purpose of Structured Analysis and Design Tools? Explain in detail.
2. Explain any two Structured Analysis and Design Tools with examples.
3. Compare DFD and ER diagrams with examples.

Lesson -7

SOFTWARE TESTING

Structure

7.0 Introduction

Software Testing is a method to check whether the actual software product matched with expected requirements and to ensure that the software product is Defect free. It involves execution of software and system components using manual and automated tools to evaluate each and every feature and aspects of software. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements. It is important because:

1. The bugs or errors in the software can be identified early and can be solved before delivery of the software product.
2. Properly tested software product ensures reliability, security and high performance.
3. Timely fixed bugs result in time saving, cost effectiveness and customer satisfaction.

Definition:

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Benefits of Software Testing: These are

- **Cost-Effective:** It is one of the important advantages of software testing. Testing any IT project on time helps to save money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix at that stage.
- **Security:** It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems at early stage of development of program.
- **Product quality:** It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
- **Customer Satisfaction:** The main aim of any product is to give satisfaction to their customers. A well-tested software built the confidence among clients that they have purchased reliable software.

7.1 Objectives of Software Testing

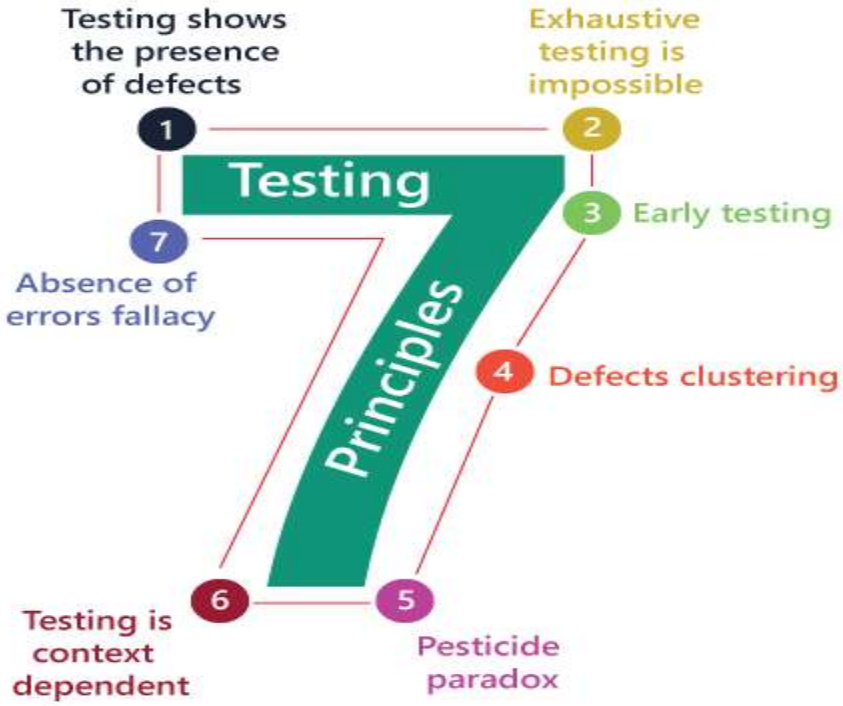
The main objectives for performing Software Testing are:

1. To ensure that the solution meets the business and user requirements; to determine user acceptability; to gain confidence that it works; evaluating the capabilities of a system to show that a system performs as intended, and verifying documentation.
2. Software Testing evaluates and maps the final software against business and user requirements. Well performed software testing will be having good test coverage. Higher the test coverage better is evaluation of the software.
3. Software testing shall design test cases with a higher probability of finding errors. The number of errors reported per defined number of test cases is quite indicative of this objective. The higher the number better the objective of the tests met.
4. Software testing shall ensure that the user accepts the final software released for him to operate with no complaints.
5. Good testing covers functionality, instability, and operational ease to learn and use the system. Also, it verifies that the system is easily deployable and replaceable.

7.2 Software Testing Principles

Software testing is a procedure of implementing software application to identify the defects and bugs. For testing an application or software, we need to follow some principles to make our product defects free.These also helps the test engineers to test the software with their effort and time.

The seven essential principles of software testing.



E. Testing shows the presence of defects

F. Exhaustive Testing is not possible, so exception handling is utilized.

G. Early Testing

H. Defect Clustering

I. Pesticide Paradox

J. Testing is context-dependent

K. Absence of errors fallacy

1. Testing shows the presence of defects:

The test engineer will test the application to make sure that the application is bug or defects free. The primary purpose of doing testing is to identify the number of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it.

2. Exhaustive Testing is not possible:

It is very hard to test all the modules and their features with effective and non- effective combinations of the inputs data throughout the actual testing process.Hence, instead of

performing the exhaustive testing as it takes boundless determinations. So, we can test the modules because the product timelines will not permit us to perform such type of testing scenarios.

3. Early Testing:

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's requirement analysis stage to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

4. Defect clustering

The defect clustering defines that throughout the testing process, we can detect the numbers of bugs that are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

5. Pesticide paradox

This principle defines that if we are executing the same set of test cases again and again over a particular time, then these kinds of test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently to accommodate market changes also. And the new and different tests are necessary to be written for the implementation of multiple parts ofs the software, which helps us to find more bugs.

6. Testing is context-dependent

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

7. Absence of errors fallacy

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested with partial understanding and interpreted the client’s requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

7.4 Software Testing Strategies

Definition:

A high-level document is used to validate the test types or levels to be executed for the product and specify the Software Development Life Cycle's testing approach is known as Test strategy document. Once the test strategy has been written, we cannot modify it, and it is approved by the Project Manager, development team.

Introduction:

In SDLC (Software Development Life Cycle), the test strategy document plays an important role. It includes various significant aspects

1. who will implement the testing.
2. what will be tested.
3. how it will be succeeded
4. what risks and incidents are concerned.

The test strategy document is approved and reviewed by the following's peoples:

L. Test Team Lead

M. Development Manager

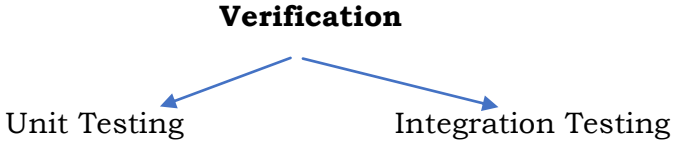
N. Quality Analyst Manager

O. Product Manager

Testing is one of the most challenging steps of the software development process. Itrequires close attention to detail. This is why software testing process is broken down into two main stages. These are verification and validation.

7.4.1 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 of user. Verification is Static Testing.



7.4.1.1 Unit Testing:

During this stage, testers evaluate individual components of the system to see if these components are functioning properly on their own or not.

For instance, if your software is based on procedural programming, these units might be individual functions or program processes. On the other hand, in object-oriented structures, these units can be in the form of a single class.

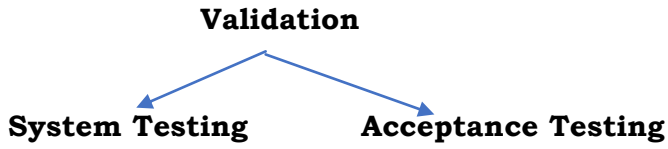
By applying unit testing in code changes, you can make sure that all issues are resolved quickly. It justifies the advantage of cohesion principle for refinement.

7.4.1.2 Integration Testing

Testers tested units as collective group. This allows to identify any problems in the interface among the modules and functions. It ensures the correct parameter passing among the units.

There are various ways to test individual components as a group, but they can differ, depending on how every single unit is defined.

1. **Validation Stage:** Validation is the process of checking whether the software product is up to the mark. It is the process of checking the validation of the product i.e., it checks what we are developing is the right product. it is a validation of actual and expected product. Validation is the Dynamic Testing.



1. **System Testing:**

System testing is the validation result of the verification process. In this stage, testers see the integrated components are performing optimally. This process is crucial for the quality life cycle, and ensure the quality standards and fulfill all major requirements.

The system testing stage is significant because it helps the application meet with its overall functional, technical, and business requirements specified by the customer.

7.4.2.2 Acceptance Testing

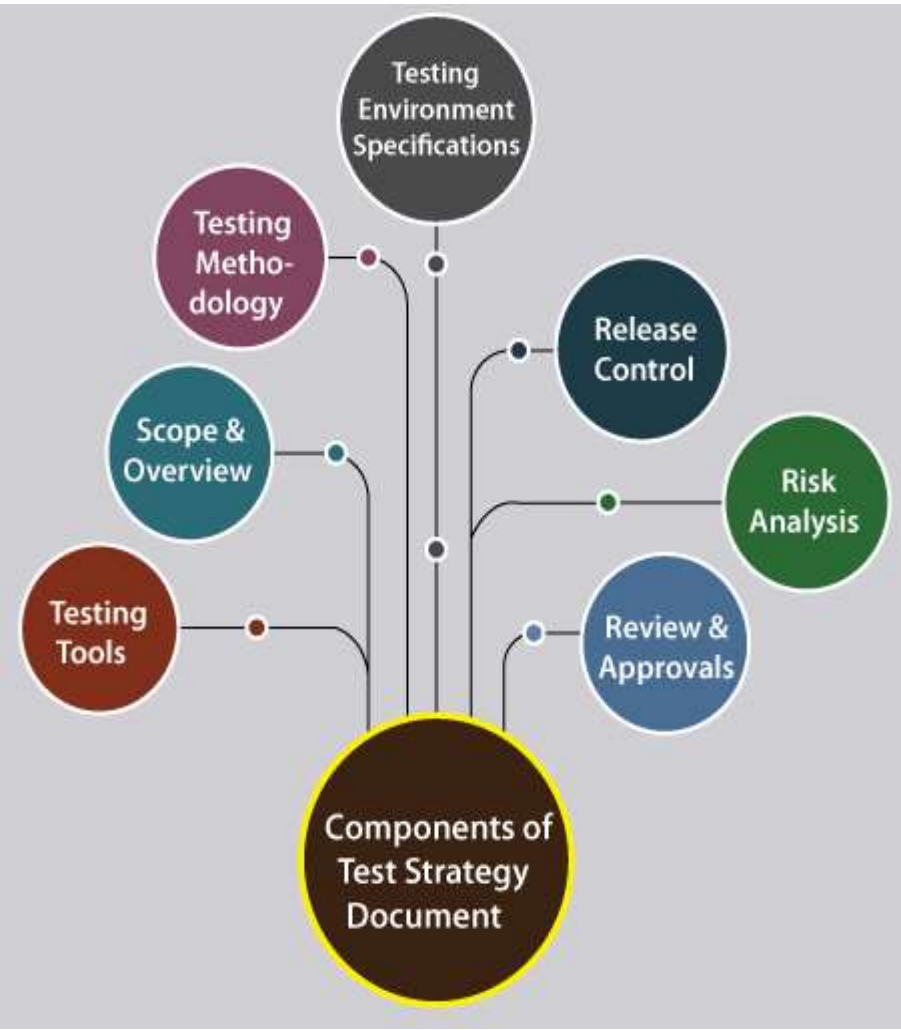
Acceptance testing is the final stage of the quality assurance (QA)test cycle.

2. It evaluates if the application is ready to be released for user consumption.
3. Testers carry out this phase with the help of the representatives of the customer who test the application by using it. It is called Beta- Testing.
4. Acceptance Testing is essential to identify any misunderstanding of business requirements and deliver the product that your customers want.
5. If this Acceptance step is ignored, it is possible that customers might not get the features they wanted and would not come back to you in the future.

Components of Test Strategy Document:

The test strategy document is made during the requirements phase and after the requirements have been listed.

Like other testing documents, the test strategy document also includes various components, such as:



1. Scope and Overview: Specifies who should approve, review and use the document, testing activities and phases needed.

2. Testing Methodology:

P. specify the levels **of testing, testing procedure, roles, and responsibilities** of all the team members.

Q. It also contains the change management and modification request submission.

3. Testing Environment Specifications: Lists

R. The information related to **the number of environments and the setup demanded**.

S. The backup and restore strategies.

4. Testing Tools: ensure the adaption of

T. **Test management** and **automation tools.**

U. The **security, performance, load testing**, and tools are **the open-source or commercial tool.**

5. Release Control:

V. Ensures the correct and effective **test execution** and release management strategies.

6. Risk Analysis:

W. All the possible risks are described linked to the project

X. Provides a contingency plan if the development team faces these risks in real-time.

7. Review and Approvals

1. All the related testing activities are reviewed and approved by: D**evelopment Team and Business Team.**

7.5 Software Testing Process: The software testing process involves the following steps.

Step-1: Assess Development Plan and Status –During this step, testers take care of completeness and correctness of event plan. In completeness of Project Plan testers can estimate quantity of resources needed for implementation of software.

Step-2:Develop the Test Plan – Test Plan is same as the software planning process. The structure of all plans should be an equivalent but content must vary.

Step-3: Test Software Requirements: Testers through verification must determine that requirements are accurate, complete and they do not conflict with one another.

Step-4: Test Software Design: This step tests both external and internal design primarily through verification techniques. The testers are concerned that planning should achieve objectives according to user requirements, so that design should be effective and efficient.

Step-5: Build phase Testing: Due to automated construction, there is less testing requirement during this phase. But if software is made using waterfall model, it is cheaper to find defects during development phase.

Step-6: Execute and Record Result: This involves testing of code to validate that executable code actually meet stated software requirements.

Step-7: Acceptance Test: Acceptance testing enables users to gauge the applicability and usefulness of software in performing their day-to-day job functions. This test ensures that software should perform as per documented requirements.

Step-8: Report Test Results –Test reporting is continuous process. It may be both oral and written. It is important that defects and concerns be reported to the appropriate parties as early as possible, so that corrections can be made at the lowest possible cost.

Step-9: The Software Installation –Once test team has confirmed that software is prepared for production use, then that software during production environment should be tested. This

tests interface to operating software, related software, and operating procedures.

Step-10: Test Software Changes –After software is implemented, there is chance of change in requirements which can directly impact on change in test plan. Hence changes on system software must be tested and evaluated.

Step-11: Evaluate Test Effectiveness: Testing can be improved by evaluating effectiveness of testing. This also involve developers, users of software and quality assurance professionals.

2. Software Testing Analysis:

Software testing analysis are the ways employed to test the application under test against the functional or non-functional requirements gathered from business. Each testing technique helps to find a specific type of defect. For example, Techniques which may find structural defects might not be able to find the defects against the end-to-end business flow. Hence, multiple testing techniques are applied in a testing project to achieve it with acceptable quality.

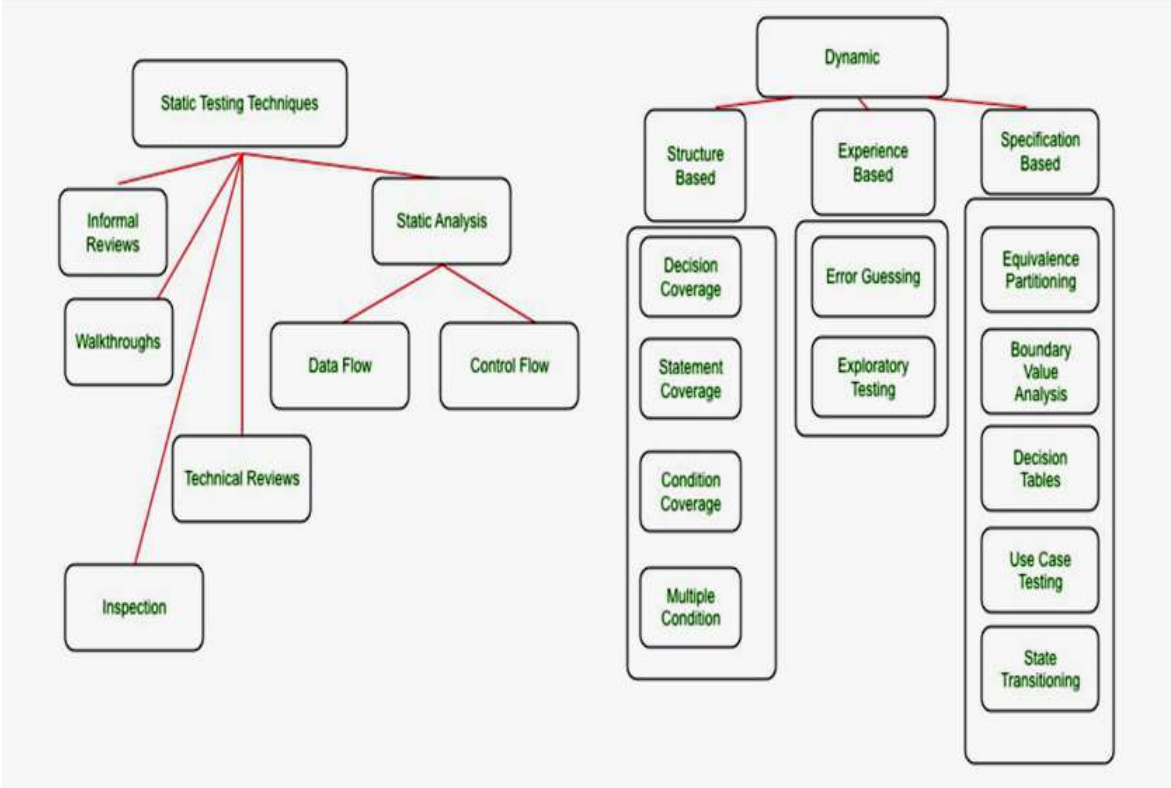
Types of Software Testing Analysis:

There are two main categories of software testing techniques:

1. **Static Analysis** are testing techniques that are used to find defects in applications under test without executing the code. Static Testing is done to avoid errors at an early stage of the development cycle and thus reducing the cost of fixing them.

2. **Dynamic Analysis** are testing techniques that are used to test the dynamic behaviour of the application under test, that is by the execution of the codebase. The main purpose of dynamic testing is to test the application with dynamic inputs- some of which may be allowed as per requirement (Positive testing) and some are not allowed (Negative Testing).

Each testing technique has further types as showcased in the below diagram.



Static Testing Analysis:

Static Testing Analysis are testing techniques that do not require the execution of a code base. Static Testing Analysis are divided into two major categories:

1. **Reviews**:
    a. Purely informal peer reviews between two developers/testers.
    b. Formal Inspections which are led by moderators who can be internal or external to the organization. Reviews are carried out through:

1.1 Peer Reviews: Informal reviews are generally conducted without any formal setup. It is between peers. For Example- Two developers review code or test cases.

1.2 Walkthroughs: Walkthrough is a category where the author of work explains the logic behind it to the stakeholders to achieve a common understanding.

1.3 Technical review: It focuses solely on the technical aspects of the document under review to achieve an agreement. It has less or no focus on the identification of defects based on reference documentation. Technical experts like architects or chief designers are required for doing the review.

1.4 Inspection: Inspection is the most formal category of reviews. Defects that are identified in the Inspection meeting are logged in the defect management tool and followed up until closure. The discussion on defects is avoided and a separate discussion phase is used, which makes Inspections a very effective form of review.

2. Static Analysis:

Static Analysis is an examination of requirement/code or design with the aim of identifying defects that may or may not cause failures. There are many tools for Static Analysis that are mainly used by developers before or during Component or Integration Testing. For example, Compiler is a Static Analysis tool as it points out incorrect usage of syntax, and it does not execute the code.

There are several aspects to the code structure – Namely Data flow, Control flow, and Data Structure.

Data Flow: It means how the data trail is followed in a given program – How data gets accessed and modified as per the instructions in the program.

Control flow: It is the structure of how program instructions get executed i.e., conditions, iterations, or loops. Control flow analysis helps to identify defects such as Dead code i.e., a code that never gets used under any condition.

Data Structure: It refers to the organization of data irrespective of code. The complexity of data structures adds to the complexity of code. Thus, it provides information on how to test the control flow and data flow in a given code.

Dynamic Testing Analysis:

Dynamic Analysis is carried through Black Box and White Box testing:

7.6 BLACK BOX TESTING

It checks the overall functionality of the complete software Project. The black box testing techniques are also called opaque testing, functional testing, close box testing, and behavioral testing.

In this type of testing the **control flow graph** is annotated with the information about how the program variables are defined and used.

Some of the advantages and disadvantages of the black-box testing technique are listed below:

Advantages

- Efficient for a large code segment.
- Tester perception is very simple.
- User's perspectives are clearly separated from developers' perspectives (programmer and tester are independent of each other).
- Quicker test case development.

Disadvantages

- Only a selected number of test scenarios are actually performed. As a result, there is only limited coverage.
- In case of partial specifications test cases are difficult to design.

Some important types of **black-box testing techniques** are briefly described below:

a. Equivalence Partitioning

1. It is generally used together and can be applied to any level of testing.
2. The idea is to partition the input range of data into valid and non-valid sections such that one partition is considered "equivalent".
3. Once we have the partitions identified, it only requires us to test with any value in a given partition assuming that all values in the partition will behave the same.
4. It can reduce the number of test cases, as it divides the input data of a software unit into a partition of data from which test cases can be derived.

For example, if the input field takes the value between 1-999, then values between 1-999 will yield similar results, and we need NOT test with each value to call the testing complete

b. Boundary Value Analysis

1. This analysis tests the boundaries of the range- both valid and invalid**.**
2. In the example above, 0,1,999, and 1000 are boundaries that can be tested. The reasoning behind this kind of testing is that more often boundaries are not handled gracefully in the code.
3. This is important technique to handle exception of Software.
4. It focuses more on testing at boundaries, or where the extreme boundary values are chosen. It includes minimum, maximum, just inside/outside boundaries, error values and typical values.

c. Cause-Effect Graph

It is a testing technique, in which testing begins by creating a graph and establishing the relation between the effect and its causes.

1. Identity, negation, logic OR, and logic AND are the four basic symbols which express the interdependency between cause and effect.
2. These are a good way to test the combination of inputs.
3. Take an example of a Credit Card that is issued if both credit score and salary limit are met. This can be illustrated in below decision table:

	Rule 1	Rule 2	Rule 3	Rule 4
Credit Score Met	FALSE	FALSE	TRUE	TRUE
Salary Limit Met	FALSE	TRUE	FALSE	TRUE
Issue Credit Card	No	No	No	Yes

d. **Use case-based Testing:** This technique helps us to identify test cases that execute the system as a whole. Use cases are a sequence of steps that describe the interaction between the user and the system. They are always defined in the language of the user, not

the system. This testing is most effective in identifying the integration defects. Use case also defines any preconditions and postconditions of the process flow.

ATM machine example can be tested via use case:

Happy	1	A: Insert Card
Flow	2	A: Enter PIN
A: Actor	3	S: Validte PIN
S: System	4	S: Allow to withdraw

Fig: Use case-based Testing

e. **State Transition Testing:** It is used where an application under test or a part of it can be treated as finite state machine. Continuing the simplified ATM example above, we can say that ATM flow has finite states and hence can be tested with the State transition technique.

There are 4 basic things to consider –

1. States a system can achieve
2. Events that cause the change of state
3. The transition from one state to other
4. Outcomes of change of state

A state event pair table can be created to derive test conditions – both positive and negative.

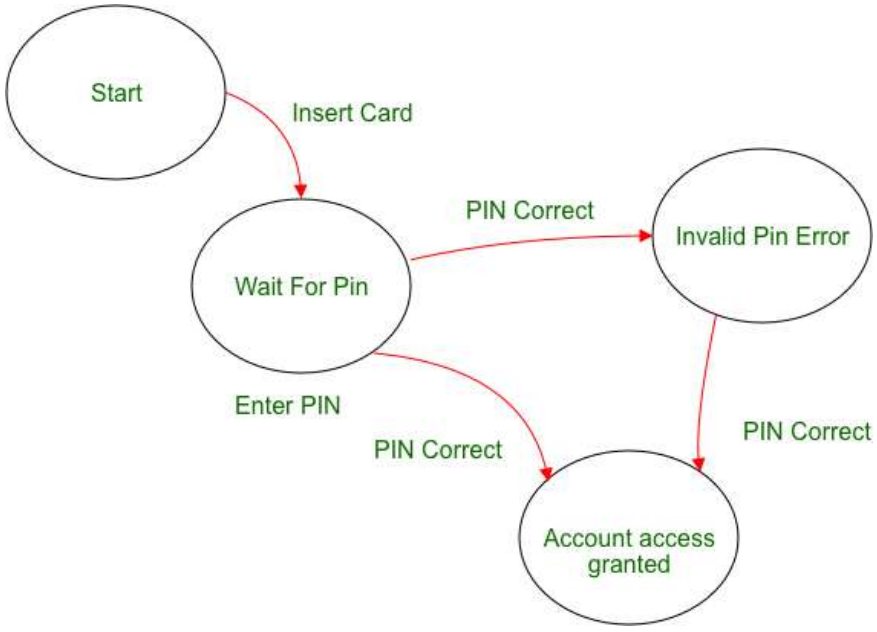


Fig: State Transition

7.7 B. WHITE BOX TESTING

White box testing is a test case design method that uses the control structure of the procedural design to derive test cases.

Features:

- White box testing can uncover implementation errors such as poor key management by analysing internal workings and structure of a software module.
- White box testing is applicable at integration, unit and system levels of the software testing process.
- In white box testing the tester needs to have a look inside the source code and find out which unit of code is behaving inappropriately.

Some of the advantages and disadvantages of white box testing technique are listed below:

Advantages

- It reveals error in hidden code by removing extra lines of code.
- Maximum coverage is attained during test scenario writing.

Disadvantages

- It is very expensive as it requires a skilled tester to perform it.
- Many paths will remain untested as it is very difficult to look into every nook and corner to find out hidden errors.

Some important **types of white box testing techniques** are briefly described below:

A. Control Flow Testing

It is a structural testing strategy that uses the program control flow as a model control flow and favours more but simpler paths over fewer but complicated path.

B. Branch Testing

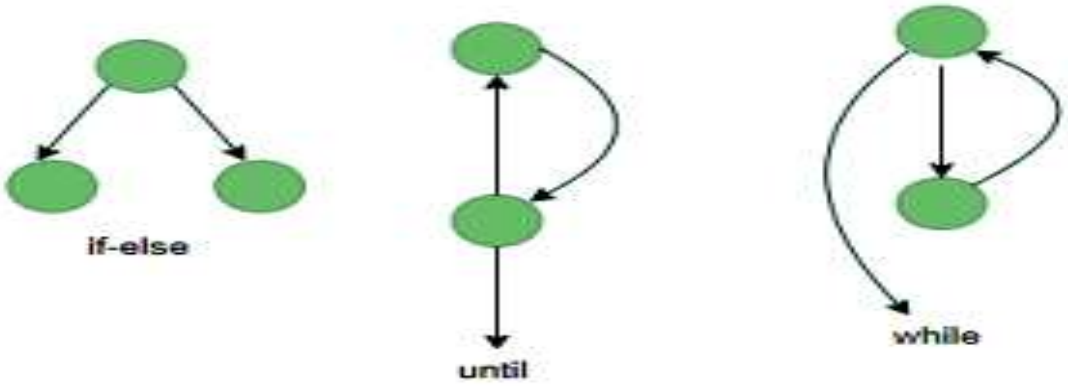
Branch testing has the objective to test every option (true or false) on every control statement which also includes compound decision.

C. Basis Path Testing

In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.
Steps:

1. Make the corresponding control flow graph
2. Calculate the cyclomatic complexity
3. Find the independent paths
4. Design test cases corresponding to each independent path

Flow graph notation: It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.

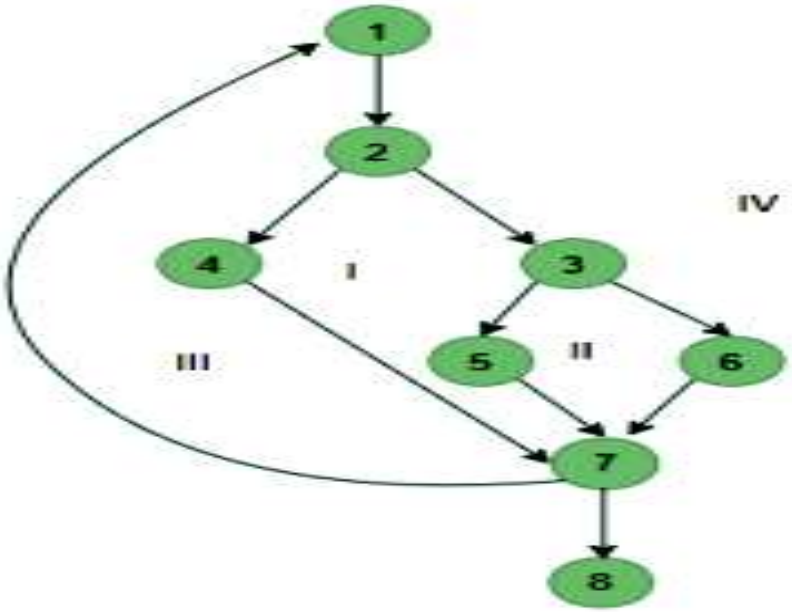


.

Cyclomatic Complexity: It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G, V(G) is its cyclomatic complexity.
Calculating V(G):

1. V(G) = P + 1, where P is the number of predicate nodes in the flow graph
2. V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
3. V(G) = Number of non-overlapping regions in the graph

Example:



V(G) = 4 (Using any of the above formulae)

No of independent paths = 4

- #P1: 1 – 2 – 4 – 7 – 8
- #P2: 1 – 2 – 3 – 5 – 7 – 8
- #P3: 1 – 2 – 3 – 6 – 7 – 8
- #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

D. Data Flow Testing

In this type of testing the control flow graph is an noted with the information about how the program variables are define and used.

E. Loop Testing

It exclusively focuses on the validity of loop construct. Errors often occur at the beginnings and ends of loops.

1. **Simple loops:** For simple loops of size n, test cases are designed that:
    - Skip the loop entirely
    - Only one pass through the loop
    - 2 passes
    - m passes, where m < n
    - n-1 and n+1 passes
2. **Nested loops:** For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
3. **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each.

   If they're not independent, treat them like nesting.

7.8 Object Oriented Testing

Introduction:

In traditional testing the parameters of the problems, the functions and their relationships tested, verified individually. Hence, it involves more user efforts. In Object Oriented Testing, all the individual variables and the associated functionality is combined into one framework known as class, which saves the computer time and space resources. Hence, reduced complexity of software**.**

Definition: Object Oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code usually called classes, which are used to create individual instances of objects**.**

Difference between Conventional Testing and Object-Oriented Testing:

Object-Oriented Testing	Conventional Testing
1. In object-oriented testing, the module or subroutine, or procedure are considered as a unit.	1. In Conventional Testing, a class is considered as a unit.
2. Here, a single operation of a procedure can be tested.	2. Here, we cannot test a single operation in isolation but rather as part of a class.
3. It focuses on composition.	3. It focuses on decomposition
4. It uses an incremental approach in the testing process.	4. It uses a sequential approach in the testing process.
5. This testing requires at every class level wherein each class is tested individually.	5. This testing is following the waterfall life cycle in its testing process.
6. This testing has a hierarchical control structure.	6. This testing does not have any hierarchical control structure.
7. Top-down or bottom-up integration is possible in this testing.	7. Here, any ordering is not possible to follow.
8. In object-oriented testing, it has unit, integration, validation, and system testing as its levels of testing.	8. Conventional Testing also has the same levels of testing but the approach is different.

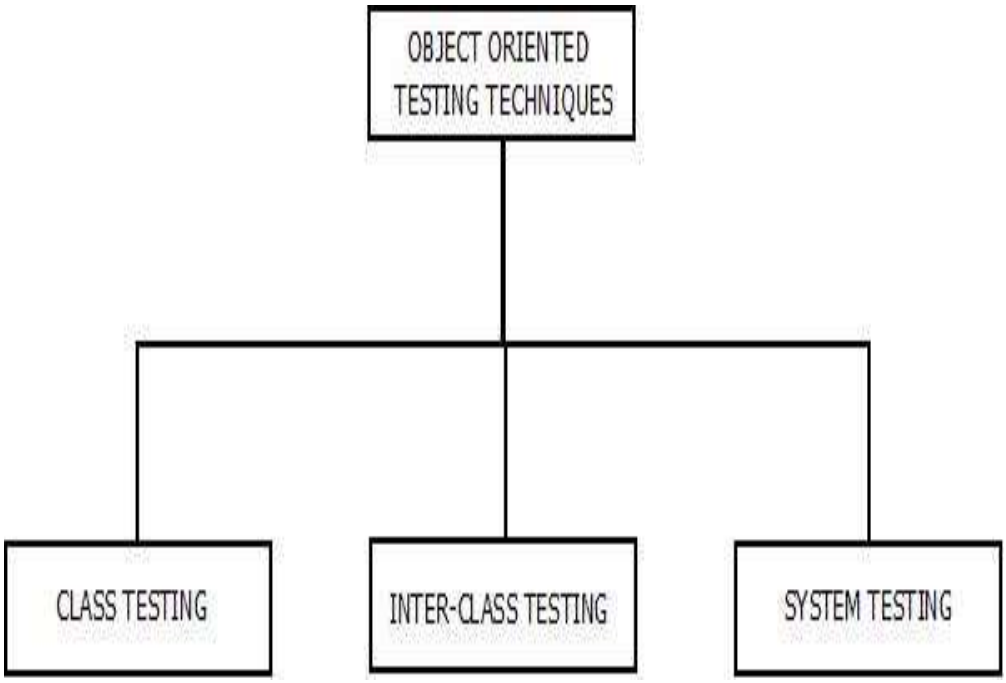
Importance:

1. It adheres with the concept of OOPs like class, object, abstraction, Inheritance etc.
2. In large-scale systems object-oriented testing is used rather than the conventional testing strategies.
3. The whole object-oriented testing revolves around the fundamental entity known as “class”.

   With the help of “class” concept, larger systems can be divided into small well-defined units which can implement separately.
4. The object-oriented testing can be classified as like conventional systems. These are called as the levels for testing.

Levels of Object-Oriented Testing:

The different levels of object-oriented testing are given as:



1. **Class Testing**
   - Class testing is also known as unit testing.
   - In class testing, every individual classes are tested for errors or bugs.
   - Class testing ensures that the attributes of class are implemented as per the design and specifications. Also, it checks whether the interfaces and methods are error free of not.
2. **Inter-Class Testing**
   - It is also called as integration or subsystem testing.
   - Inter class testing involves the testing of modules or sub-systems and their coordination with other modules.
3. **System Testing**
   - In system testing, the system is tested as whole and primarily functional testing techniques are used to test the system. Non-functional requirements like **performance, reliability, usability and test-ability are also tested.**

Techniques of Object-Oriented Testing:

1. Fault Based Testing:

This type of checking permits the test cases supported the consumer specification or the code or both. It tries to identify possible faults (areas of design or code that

may lead to errors.). For all of these faults, a test case is developed to "flush" the errors out. These tests also force each time of code to be executed.

This method of testing does not find all types of errors. However, incorrect specification and interface errors can be missed. These types of errors can be uncovered by function testing in the traditional testing model. In the object-oriented model, interaction errors can be uncovered by scenario-based testing. This form of Object oriented-testing can only test against the client's specifications, so interface errors are still missed.

2. **Class Testing Based on Method Testing:**

This approach is the simplest approach to test classes. Each method of the class performs a well-defined cohesive function and related to unit testing of the traditional testing techniques. Therefore, all the methods of a class can be involved at least once to test the class.

3. **Random Testing:**

It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behaviour of the categories.

4. **Partition Testing:**

This methodology categorizes the inputs and outputs of a category so as to check them severely. This minimizes the number of cases that have to be designed.

5. **Scenario-based Testing:**

It primarily involves capturing the user actions then stimulating them to similar actions throughout the test. These tests tend to search out interaction form of error.

Example

- **A bank account class**.
  - It may have methods to open the account, deposit funds, withdraw funds and close the account.
- **The states of the account include**
  - open with positive balance,
  - open with negative or zero balance and
  - closed.
- **How the methods behave depends on the state of the account.**
  - An account with a zero or negative balance will not allow the customer to withdraw funds.
  - If positive it might allow customer to go to overdraft once.
  - You could introduce a new method to determine the if the account is open or closed and if balance is positive.

Limitations:

1. Classes obvious unit choice, but they can be large in some applications
2. Problems dealing with polymorphism and inheritance

7.9 Summary

In this chapter, we have studied about software testing and its process. The different white box and black box techniques are explored for their advantages and disadvantages.

7.10 Suggested Readings

1. "Software Engineering: A Practitioner's Approach" 5th Ed. by Roger S. Pressman, Mc-Graw-Hill.
2. "Software Engineering" by Ian Sommerville, Addison-Wesley, 7th Edition.
3. Software Engineering: A Practitioner's Approach, by Pressman, R. (2003), New York: McGraw-Hill.

7.11 Model Questions

1. List various testing objectives.
2. Explain testing process.
3. Draw difference between Black box and white box testing with examples.

Lesson-8

SOFTWARE QUALITY AND MAINTENANCE

Structure

8.0 Objectives

Software development efforts result in the delivery of a software product that satisfies user requirements. Accordingly, the software product must change or evolve. Once in operation, defects are uncovered, operating environments change, and new user requirements surface. The maintenance phase of the life cycle begins with a warranty period or post-implementation support delivery, but maintenance activities occur much earlier.

8.1 Introduction

Software maintenance is an integral part of a software life cycle. However, it has not received the same degree of attention that the other phases have. Historically, software development has had a much higher profile than software maintenance in most organizations. This is now changing, as organizations strive to squeeze the most out of their software development investment by keeping software operating as long as possible. The open-source paradigm has brought further attention to the issue of maintaining software artifacts developed by others.

Software maintenance is defined as the totality of activities required to provide cost-effective support to software. Activities are performed during the pre-delivery stage as well as during the post-delivery stage. Pre-delivery activities include planning for post-delivery operations, maintainability, and logistics determination for transition activities. Post-delivery activities include software modification, training, and operating or interfacing to a help desk.

8.2 SOFTWARE QUALITY ATTRIBUTES

Quality attributes are the overall factors that affect run-time behavior, system design, and user experience. They represent areas of concern that have the potential for application-wide impact across layers and tiers. Some of these attributes are related to the overall system design, while others are specific to run time, design time, or user centric issues. The extent to which the application possesses a desired combination of quality attributes such as usability, performance, reliability, and security indicates the success of the design and the overall quality of the software application.

When designing applications to meet any of the quality attributes requirements, it is necessary to consider the potential impact on other requirements. You must analyze the trade-offs between multiple quality attributes. The importance or priority of each quality attribute differs from system to system; for example, interoperability will often be less important in a single use packaged retail application than in a line of business (LOB) system.

Category	**Quality attribute**	**Description**
Design Qualities	*Conceptual Integrity*	Conceptual integrity defines the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as factors such as coding style and variable naming.
	Maintainability	Maintainability is the ability of the system to undergo changes with a degree of ease. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements.
	Reusability	Reusability defines the capability of components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components and also the implementation time.
Run-time Qualities	*Availability*	Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load.

	Interoperability	Interoperability is the ability of different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. An interoperable system makes it easier to exchange and reuse information internally as well as externally.
	Manageability	Manageability defines how easy it is for system administrators to manage the application. The sufficient and useful resources are fully utilized in monitoring systems for debugging and performance tuning.
	Performance	Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time.
	Reliability	Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.
	Scalability	Scalability is the ability of a system to either handle increases in load without impact on the performance of the system.
	Security	Security is the capability of a system to prevent malicious or accidental action sand to prevent disclosure or loss of information. A secure system aims to protect assets and prevent unauthorized modification of information.
System Qualities	*Supportability*	Supportability is the ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.

	Testability	Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner.
User Qualities	*Usability*	Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience.

8.3 Software Quality Standards:

The mission of the International Standard Organization is to market the development of standardization and its related activities to facilitate the international exchange of products and services. It also helps to develop cooperation within the different activities like spheres of intellectual, scientific, technological, and economic activity.

ISO 9000 Quality Standards:

It is defined as the quality assurance system in which quality components can be organizational structure, responsibilities, procedures, processes, and resources for implementing quality management. Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.

Different types of ISO standards:

There are different types of ISO standards as follows.

ISO 9000: 2000 –

ISO 9000: 2000: contains Quality management systems, fundamentals, and vocabulary.

ISO 9000-1: 1994 –

This series of standards includes Quality management systems and Quality assurance standards. It also includes some guidelines for selection and use.

ISO 9000-2: 1997 –

This series of standards also includes Quality management systems and Quality assurance standards. It also includes some guidelines for the application of ISO 9001, ISO 9002, and ISO 9003.

ISO 9000-3: 1997 –

This series contains Quality management systems, Quality assurance standards and also includes guidelines for the application of ISO 9001 to 1994 to the development, supply, installation, and maintenance of computer installation.

ISO 9001: 1994 –

This series of standards has Quality systems and a Quality assurance model. This model helps in design, development, production, installation, and service.

ISO 9001: 2000 –

This series of standards also includes Quality management systems.

ISO 9002: 1994 –

This series of standards also includes some Quality systems. This Quality assurance model used in production, installation, and servicing.

ISO 9003: 1994 –

This series of standards also includes some Quality systems. This Quality assurance model used in the final inspection and test.

ISO 9004: 2000 –

This series of standards include some Quality management systems. It also includes some guidelines for performance improvements.

ISO 9039: 1994 –

This series of standards include some Optics and Optical Instruments. It includes quality evaluation of optical systems and determination of distortion.

ISO/IEC 9126-1: 2001 –

This series of standards has information technology. It also includes some software products, quality models.

ISO/IEC 9040: 1997 –

This series of standards has information technology. It also includes open system interconnection and Virtual terminal basic class service.

ISO/IEC 9041-1: 1997 –

This series of standards has information technology. It also includes open system interconnection, Virtual terminal basic class service protocol, and specification.

ISO/IEC 9041-2: 1997 –

This series of standards include information technology, open system interconnection, Virtual terminal basic class protocol, and Protocol implementation conformance statement (PICS) proforma.

ISO/IEC 9075-9: 2001 –

This series of standards has information technology, Database languages, and SQL/MED(Management of External Data).

ISO/IEC 9075-10: 2000 -|

This series of standards has information technology, Database languages, and SQL/OLB (Object Language Bindings).

ISO/IEC 9075-13: 2002 –

This series of standards has information technology, Database languages, SQL routines, and Java Programming language. (SQL/JRT).

8.4 FACTORS AFFECTING SOFTWARE QUALITY

A software quality factor is a non-functional requirement for a software program that enhances the quality of the software program. Some software quality factors are listed here:

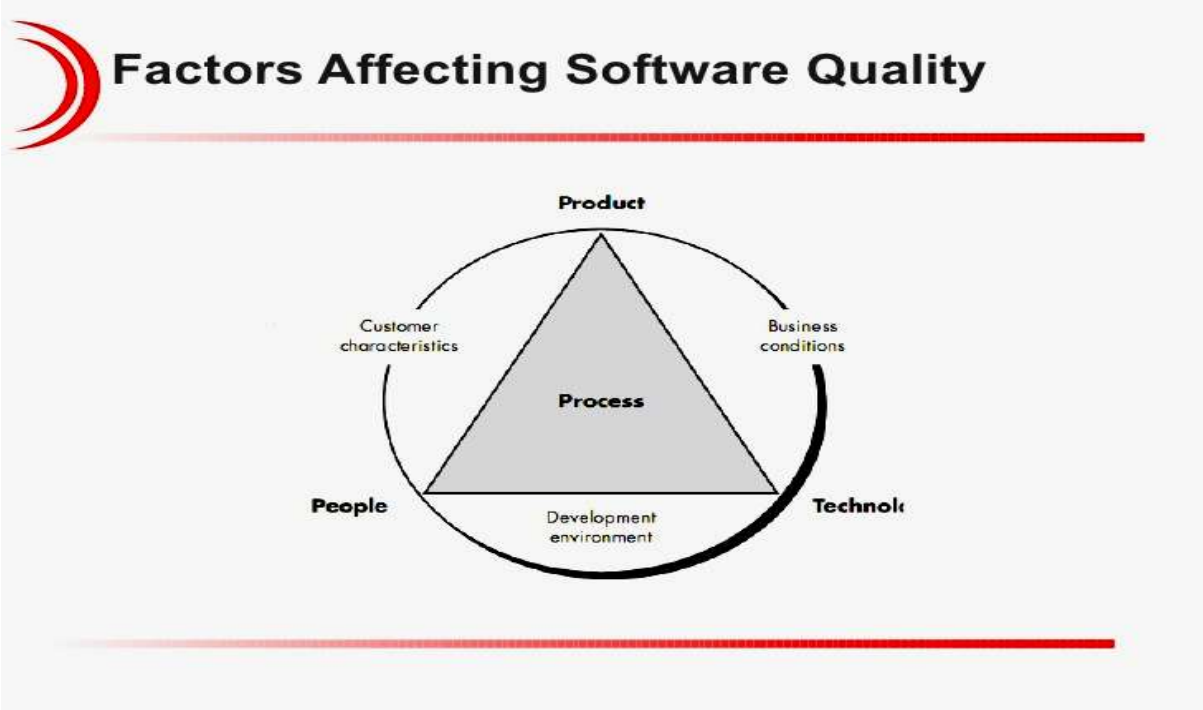


Figure1: Factors affecting Software Quality

1. Understandability

Clarity of purpose: All of the design and user documentation must be clearly written so that it is easily understandable. This is obviously subjective in that the user context must be taken into account: for instance, if the software product is to be used by software engineers it is not required to be understandable to the layman.

2. Completeness

Presence of all constituent parts, with each part fully developed. This means that if the code calls a subroutine from an external library, the software package must

provide reference to that library and all required parameters must be passed. All required input data must also be available.

3. Conciseness

Minimization of excessive or redundant information or processing: This is important where memory capacity is limited, and it is generally considered good practice to keep lines of code to a minimum. It can be improved by replacing repeated functionality by one subroutine or function which achieves that functionality. It also applies to documents.

4. Portability

It is the ability to be run well and easily on multiple different computer configurations. Portability can mean both between different hardware—such as running on a PC as well as a smart phone—and between different operating systems—such as running on both Mac OS X and GNU/Linux.

5. Maintainability

Propensity to facilitate updates to satisfy new requirements. Thus, the software product that is maintainable should be well-documented, should not be complex, and should have spare capacity for memory, storage and processor utilization and other resources.

6. Testability

To justify acceptance criteria and evaluation of system performance: Such a characteristic must be built-in during the design phase to make the product easily testable; a complex design leads to poor testability.

7. Usability

Convenience and practicality of use: This is affected by such things as the human-computer interface. The component of the software that has most impact on this is the user interface (UI), which for best usability is usually graphical Interface (GUI).

8. Reliability

Expected ability to perform its intended functions satisfactorily. This implies that a reliable product is expected to perform correctly over a period of time. It also encompasses environmental considerations in that the product is required to perform correctly in whatever conditions it finds itself (sometimes termed robustness).

9. Efficiency

Fulfilment of purpose without waste of resources, such as memory, space and processor utilization, network bandwidth, time. All modules and their relationships are validated effectively.

10. Security: Ability to protect data against unauthorized access to different operations. Besides the presence of appropriate security mechanisms such as authentication, access control and encryption, security specifies the various firewall and other security shields to handle malicious, intelligent and adaptive attackers.

8.5 AIMS OF SOFTWARE MAINTENANCE

Software maintenance is required due to huge cost and slow speed of system. A software that is useful and successful user-generated requests for change and improvements.

c. In order to improve the speed and accuracy of change while reducing costs, key problems and promising solution strategies are identified. The aims are met, by taking two approaches.

d. Current trends and practices are projected forward using a model of software evolution

e. Various technical and business facets of maintenance (e.g. risk models) are studied to verified their long term effect on the product. For example, Lehman is using feedback control models in the FEAST project.

To understand how such models may be exploited in an industrial context (for example, in cost estimation). This work leads to better metrics.

f. It is a metric to indicate the importance of project in Industrial Context.

g. To establish accepted evaluation procedures for assessing new added developments and processes due to changes in technology.

8.6 TYPES OF SOFTWARE MAINTENANCE

In a software lifetime, the type of maintenance may vary based on its nature. It may be just a routine maintenance task as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by the user or concluded by user error reports.

- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to-date and tuned to the ever-changing world of technology and business environment.

- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improving its reliability and performance.

- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

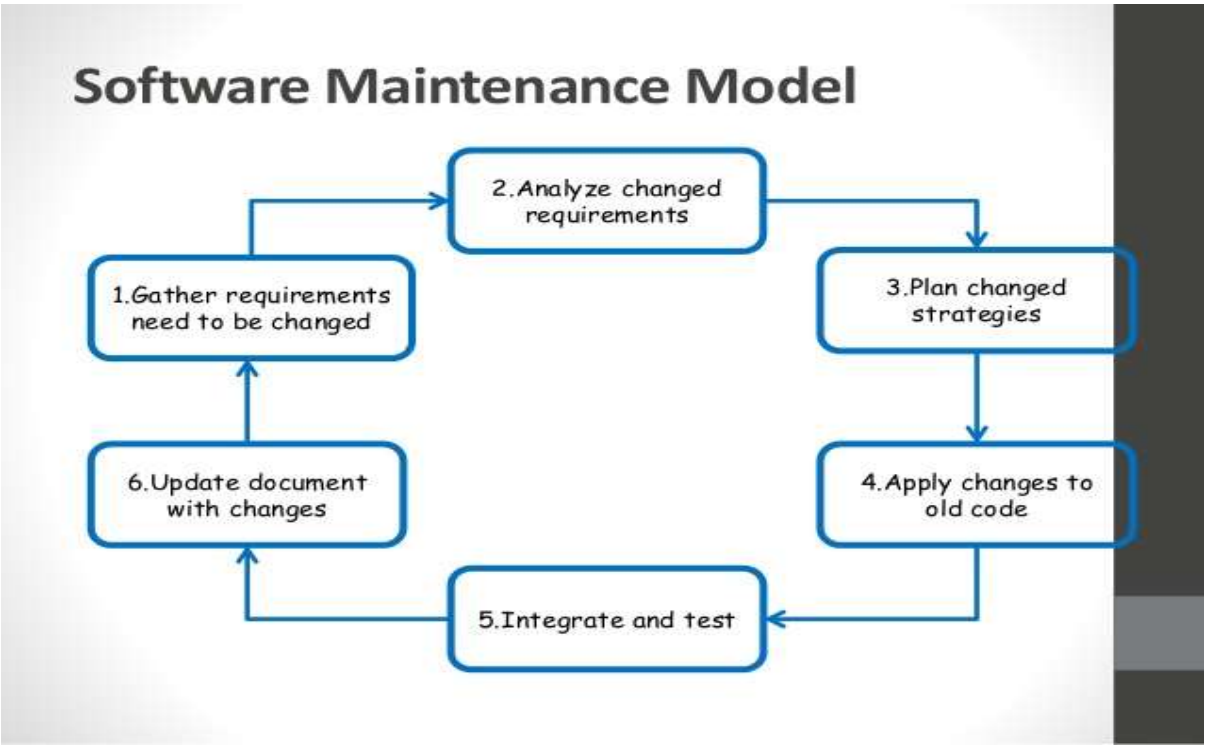
8.7 Maintenance Activities and Costs

IEEE provides a framework for sequential maintenance process activities. It can be used in an iterative manner and can be extended so that customized items and processes can be included. Various activities for different phases are:

A. **Identification & Tracing** - It involves activities pertaining to the identification of requirements to be modified and maintained. It is generated by users and software engineers.

B. **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for.

C. **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.



D. **Implementation** - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.

E. **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

F. **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

G. **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered. Training facility is provided if required, in addition to the hard copy of user manual.

H. **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

8.8 SOFTWARE MAINTENANCE COSTS:

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Need for Maintenance –

Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.

Software maintenance cost factors:

The key factors that distinguish development and maintenance and which lead to higher maintenance cost are divided into two subcategories:

1. Non-Technical factors
2. Technical factors

Non-Technical factors:

The Non-Technical factors include:

1. Application Domain
2. Staff stability
3. Program lifetime
4. Dependence on External Environment
5. Hardware stability

Technical factors:

Technical factors include the following:

1. module independence
2. Programming language
3. Programming style

4. Program validation and testing
5. Documentation
6. Configuration management techniques

Efforts expanded on maintenance may be divided into productivity activities (for example analysis and evaluation, design and modification, coding). The following expression provides a module of maintenance efforts:

$M = P + K(C - D)$

where,

M: Total effort expanded on the maintenance.

P: Productive effort.

K: An empirical constant.

C: A measure of complexity that can be attributed to a lack of good design and documentation.

D: A measure of the degree of familiarity with the software.

For example: A mobile app to maintain. Let's try to calculate the approximate budget.

- An app hosting server can cost anywhere from $70 per month up to $320 per month, which largely depends on such factors as the app's content, the number of active users, and its projected growth;
- The fees for iOS and Android accounts are 99$ and 25$ respectively;
- Software licenses can be free (say, for using iTunes) or cost up to tens of thousands of dollars (say, for using EDA tools);
- SaaS account fees can be up to a thousand dollars;
- Let's say your developers' rate is 55$/hour and the time needed to eliminate one bug is 1 hour. If the average amount of bugs per month is 20, you will need to spend 1100$ on debugging;
- The average hourly rate of a designer is 25$. For example, a quick design update requires about half a week of work. So, designer services will cost you 500$ a month;
- There is no need for code refactoring because everything is ready long before the app's release. But in reality, there are situations when developers have to write code very quickly. As a result, the code often ends up being pretty messy, so it requires some refactoring. A developer needs up to 3 hours to finish this task. According to the mentioned rates, it will cost about 150$.
- In fact, there are so many possible unforeseen costs, that it's really hard to consider all of them. There are several main factors that make up the total software maintenance cost. Let's make some calculations with those factors.

The key factors which have a great impact on software maintenance cost are:

Maintenance specialists	35%
Low complexity of base code	32%
Re-engineering tools	27%
High level programming languages	25%
Reverse engineering tools	23%
Complexity analysis tools	20%
Defect tracking tools	20%
Quality measurements	16%
Formal base code inspections	15%

8.9 Summary

In this chapter we have studied the concept of software quality in terms of their attributes and factors which can influence the quality. Various issues regarding maintenance are also discussed in detail to deliver efficient software which may be reviewed for the future.