GAUHATI UNIVERSITY
Institute of Distance and Open Learning

Semester- II

MSc-IT
Paper: INF 2036

Software Engineering

www.idolgu.in

GAUHATI UNIVERSITY
Institute of Distance and Open Learning

Second Semester
(under CBCS)

M.Sc.-IT

Paper: INF-2036

SOFTWARE ENGINEERING

Contents:

BLOCK I: FUNDAMENTALS OF SOFTWARE ENGINEERING AND SOFTWARE DESIGN

BLOCK II: SOFTWARE CODING, TESTING AND MAINTENANCE

BLOCK III: SOFTWARE RELIABILITY AND SOFTWARE MANAGEMENT

Contributors:

Mr. Hem Chandra Das (Block I : Units- 1 & 2)
Asstt. Prof., Dept. of Computer
Science & Technology
Bodoland University
Kokrajhar (BTAD), Assam

Mrs. Chayanika Talukdar (Block I: Units- 3 & 4)
Asstt. Prof., Dept. of Computer Science
NERIM, Guwahati, Assam

Mrs. Shilpi Singh (Block I : Unit- 5, Block II: Units- 1 &2)
Asstt. Prof., Dept. of Computer Science
LCB College, Guwahati, Assam

Dr. Gautam Chakrabarty (Block I : Units- 6 & 7)
Asstt. Prof., Dept. of Computer Science
NERIM, Guwahati, Assam

Mr. Subrat Chetia (Block I: Unit- 8)
Asstt. Prof., Dept. of Computer Science
PDUAM, Dalgaon,Assam

Mr. Debashis Dev Misra (Block II: Unit- 3)
Asstt. Prof., Dept. of Computer Science and Engineering
Royal Global University, Guwahati, Assam

Mrs. Pinky Saikia Dutta (Block II : Unit- 4)
Asstt. Prof., Dept. of Computer Science and Engineering
GIMT, Guwahati, Assam

Dr. Utpal Barman (Block II: Unit- 5)
Asstt. Prof., Dept. of Computer Science and Engineering
GIMT, Guwahati, Assam

Dr. Aniruddha Deka (Block III: Unit- 1)
Asstt. Prof., Dept. of Computer Science and Engineering
Royal Global University, Guwahati, Assam

Mr. Adarsh Pradhan (Block III: Units- 2 & 3)
Asstt. Prof., Dept. of Computer Science and Engineering
GIMT, Guwahati, Assam

Content Editor:

Prof. Shikhar Kumar Sarma
Head, Dept. of Information Technology, Gauhati University

Course Coordination:

Prof. Dandadhar Sarma Director, IDOL, Gauhati University
Prof. Anjana Kakoti Mahanta Prof., Dept. Computer Science, G.U.

Cover Page Designing:

Bhaskar Jyoti Goswami IDOL, Gauhati University

May, 2022

BLOCK I:
FUNDAMENTALS OF SOFTWARE ENGINEERING AND SOFTWARE DESIGN

UNIT 1: INTRODUCTION TO SOFTWARE ENGINEERING

Unit Structure:

1.1 INTRODUCTION

Software Engineering is a study and approach to the design, development, operation, and maintenance of software systems that is methodical, disciplined, and quantitative.

1.2 UNIT OBJECTIVES

Software engineering's main objective is to create software development processes and procedures that can scale up for large systems and can be utilized consistently to produce high-quality

software at a low cost and with a short cycle time. In this chapter we discussed the characteristics of software, Classification of software, software crisis, about software, software engineering, approach to software engineering, software engineering challenges and software development life cycle.

1.3 SOFTWARE

A programme or set of programmes containing instructions that offer desired functionality is referred to as software. And engineering is the process of creating something that fulfils a specific function and solves problems in a cost-effective manner.

1.4 CHARACTERISTICS OF SOFTWARE

- Maintainability – The programme should be able to evolve to suit changing requirements.

- Efficiency – Software should not waste computational resources such as memory, CPU cycles, and so on.

- Correctness – If the various requirements mentioned in the SRS document have been correctly implemented, a software product is correct.

- Reusability – If the various modules of a software product can be easily reused to construct other products, the product has strong reusability.

- Testability – In this case, software aids in the creation of test criteria as well as the evaluation of the software against those requirements.

- Reliability – It's a criterion for software quality. Over an indeterminate time period, the extent to which software may be expected to accomplish its desired function.

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

- Adaptability– In this situation, the programme supports a variety of system constraints, and the user's needs can be met by altering the software.

- Interoperability – The ability of two or more functional units to work together to process data.

1.5 CLASSIFICATION OF SOFTWARE

- System Software – System software is required to manage computer resources and facilitate application programme execution. This category includes operating systems, assemblers, compilers, editors, drivers, linkers and loaders etc. System software is required for the operation of a computer. Operating systems controls the memory and operations of the computer, as well as all of its software and hardware. The compiler translates the programmer's source code (high-level language) into a machine-level language (low-level language). Assembler is a programme that translates assembly code(low-level language) into machine code(target code).

- Networking and Web Applications Software – Computer networking software offers the necessary functionality for computers to communicate with one another and with data storage facilities. When software is operating on a network of computers, networking software is also used (such as World Wide Web). It comprises all network administration software, server software, security and encryption software, and web-based application development tools such as HTML, PHP, and XML, etc.

- Embedded Software – This sort of software is embedded in the hardware, usually in the Read Only Memory (ROM), as part of a larger system, and is used to support specific functions under the control conditions. Software used in instrumentation and control applications such as washing machines, satellites, microwaves, and so on.

- Reservation Software – A reservation system is generally used to store and retrieve information about air travel, vehicle rentals, hotels, and other activities, as well as to conduct transactions. They also provide access to bus and train reservations; however they aren't always linked to the main system. These are also used in the hotel business to communicate computerised information to users, such as

making a reservation and making sure the hotel is not overbooked.

- Business Software – This type of software is used to support business applications and is the most common type of software. Inventory management, accounting, banking, hospitals, schools, stock markets, and other software are examples.

- Entertainment Software – Education and entertainment software is a valuable tool for educational organisations, particularly those who work with young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.

- Artificial Intelligence Software – Expert systems, decision support systems, pattern recognition software, artificial neural networks, and other types of software are included in this area. Complex problems are involved, and complex computations using non-numerical algorithms have no impact.

- Scientific Software – Scientific and engineering software supports a scientific or engineering user's requirements for performing enterprise-specific tasks. This type of software is created for specific applications using industry-specific principles, techniques, and formulae. Software such as PYTHON, MATLAB, AUTOCAD, PSPICE, ORCAD, and others are examples.

- Utilities Software – These programmes execute specific jobs and differ in size, cost, and complexity from other software. Anti-virus software, speech recognition software, compression programmes, and other programmes are examples.

- Document Management Software – To reduce paperwork, Document Management Software is used to track, manage, and store papers. Such systems can maintain track of all the many versions created and edited by different users (history tracking). Storage, versioning, metadata, security, as well as indexing and retrieval are all typical features.
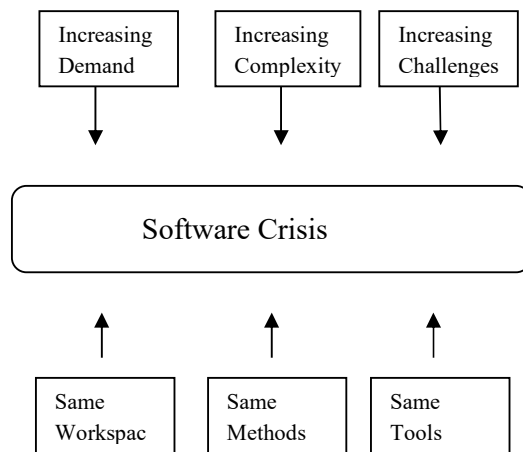
1.6   SOFTWARE CRISIS

The problem of building meaningful and efficient computer programmes in the time allotted is referred to as a software crisis in

computer science. Despite rapidly expanding software demand, complexity of software, and software issues, the software crisis was caused by the use of the same workforce, methodologies, and tools. With increase in the complexity of software, many software problems arise because existing methods were insufficient.

If we continue to employ the same workforce, processes, and tools in the face of rapidly expanding software demand, complexity, and challenges, we will face issues such as software Size and Cost, software budgeting, software efficiency, software quality, and software managing and delivering, and so on. This is referred to as a "software crisis."

```
┌──────────┐   ┌──────────┐   ┌──────────┐
│Increasing│   │Increasing│   │Increasing│
│ Demand   │   │Complexity│   │Challenges│
└────┬─────┘   └────┬─────┘   └────┬─────┘
     │              │              │
     ▼              ▼              ▼
┌─────────────────────────────────────────┐
│            Software Crisis               │
└─────────────────────────────────────────┘
     ▲              ▲              ▲
     │              │              │
┌─────────┐   ┌─────────┐   ┌─────────┐
│  Same   │   │  Same   │   │  Same   │
│Workspac │   │ Methods │   │  Tools  │
└─────────┘   └─────────┘   └─────────┘
```

1.6.1 Causes of Software Crisis

- Software's complexity and expectations are increasing on a daily basis. Software is becoming more expensive and more complex.

- The cost of owning and maintaining software was equal to the cost of creating it.

- At the time, Projects were running late.

- Software was inefficient at the time.

- The software's quality was poor.

- Software frequently failed to fulfil criteria.

- The average software project is half an hour behind schedule.

- Software was never delivered at the time.

1.6.2 Solution of Software Crisis

The crisis does not have a single solution. One possible solution of software crisis is Software Engineering because software engineering is a systematic, disciplined and quantifiable approach. There are certain measures to follow in order to avoid a software crisis:

- Software cost overruns are reduced.

- Software must be of excellent quality.

- Software development takes less time.

- Previous experience as a member of a software development team.

- Software must be made available.

1.7   SOFTWARE ENGINEERING

Software engineering refers to the application of systematic engineering principles to the creation of software products and applications. It is a discipline of engineering concerned with the analysis of user requirements, software design, development, testing, and maintenance. Software engineering results in a product that is both efficient and reliable.

1.8 APPROACHES TO SOFTWARE ENGINEERING

The following are some fundamental principles of good software engineering:

- Better Requirement Analysis is a fundamental software engineering technique that provides a clear picture of the project. Finally, by producing a good software product that meets user requirements, a thorough understanding of customer requirements adds value to its consumers.

- The KISS (Keep it Simple, Stupid) philosophy should be followed in all designs and implementations. It simplifies the code, making debugging and maintenance becomes simple.

- 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.

- Much functionality is included in software projects; all functionalities should be designed using a modular approach to make development faster and easier. Because of this modularity, functions or system components are self-contained.

- Abstraction is a specialisation of the idea of separation of concerns for suppressing complex things and offering simplicity to the customer/user, which means it provides only what the user need and hides the rest.

- Consider this. Act is a must required principle in software engineering, which says that before beginning to develop functionality, you must first consider application architecture, as a well-planned project development flow yields superior results.

- When a developer combines all features, he or she may subsequently discover that they are no longer needed. As a result, adhering to the Never Add Extra approach is critical because it implements just what is actually required, saving time and effort.

- When other developers work on another developer's code, they should not be startled and should not waste their time trying to figure out what's going on. As a result, improving documentation at critical stages is an excellent method to develop software projects.

- The Law of Demeter should be obeyed since it separates classes based on their functionality and decreases coupling (connections and interdependence between classes).

- The developers should design the project in such a way that it satisfies the principle of generality, which means that it should not be limited or restricted to a specific set of cases/functions, but rather should be free of unnatural constraints and capable of providing comprehensive service to customers who have specific or general needs.

- The principle of consistency is significant in coding style and GUI (Graphical User Interface) design because consistent

coding style makes code easier to read and consistent GUI makes user learning of the interface and software easier.

- Never waste time if something is needed but it is already out of the way; instead, using open source to fix it in your own way.

- Continuous validation ensures that a software system meets its requirements and serves its intended function, resulting in improved software quality control.

- To get out of the current technological market To meet users' needs in the most up-to-date and progressive method, it's critical to employ modern programming practises.

- To grow and handle rising demand for software applications, scalability in software engineering should be maintained.

1.9 SOFTWARE ENGINEERING CHALLENGES

The first stage in the Requirement Engineering process is to collect requirements. It assists the analyst in gaining understanding of the problem domain, which is then utilised to create a formal software specification. During this procedure, there are a variety of issues and challenges that must be overcome. The following are a few of them:

- Understanding large and complex system requirements is difficult –]

   The term " large" has two meanings:

   (i) Due to the enormous number of users, there are significant security and other constraints.

   (ii) There will be a large number of functions to implement.
- Undefined system boundaries – There might be no defined set of implementation requirements. The customer may go on to include several unrelated and unnecessary functions besides the important ones, resulting in an extremely large implementation cost which may exceed the decided budget.
- Customers/Stakeholders are not clear about their needs – Customers themselves may be unsure about the comprehensive list of functionalities they want in the software

at times. This can happen when people have a general notion of what they want but haven't thought much about how to get it done.

- Conflicting requirements are there –There's a chance that two different project stakeholders will make demands that are incompatible with each other's implementation. In some cases, a single stakeholder may articulate two requests that are mutually exclusive.

- Changing requirements is another issue – If the client communicates a modification in the first set of specified criteria through subsequent interviews or evaluations, it is possible that the customer will alter their mind. While certain criteria are simple to meet, dealing with constantly changing requirements can be tough.

- Partitioning the system suitably to reduce complexity – Sometimes projects are divided down into smaller modules or functionalities, which are subsequently handled by distinct teams. More division is required for more sophisticated and large projects. It is necessary to guarantee that the divisions are non-overlapping and self-contained.

- Validating and Tracing requirements – It is critical to double-check the mentioned requirements before beginning the implementation phase.Also, there should be forward as well as backward traceability. For eg, all the entity names should be the same everywhere, i.e., There should never be a situation where the terms "EMLOYEE" and "EMPLOYEES" are used interchangeably to refer to the same entity.

- Identifying critical requirements – It's critical to identify the set of requirements that must be implemented at all costs. The requirements should be prioritised so that the most important ones can be implemented first and foremost.

- Resolving the "to be determined" part of the requirements – Those needs that have yet to be resolved in the future are included in the TBD set of requirements. It's best to keep the number of such requirements as low as possible.

- Proper documentation, proper meeting time and budget constraints – Maintaining adequate documentation is a constant problem, especially when requirements change. Time and budget constraints must also be carefully and methodically managed.
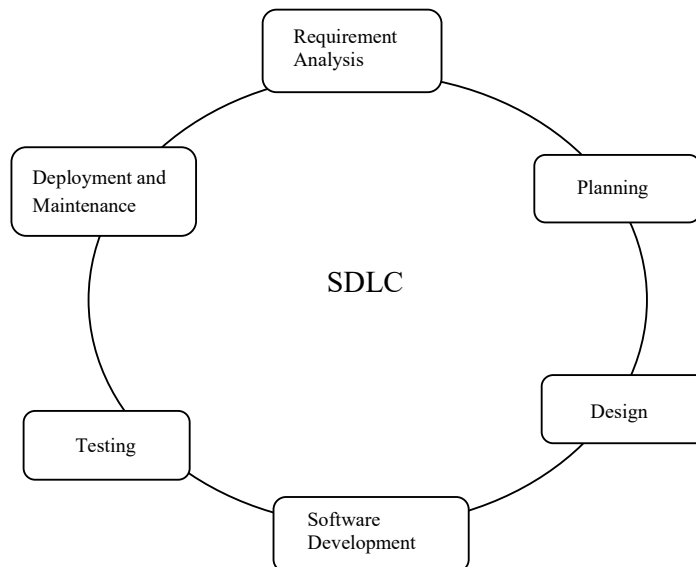
1.10 SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

The Software Development Life Cycle (SDLC) is a methodology for producing high-quality software that follows a set of well defined processes. It is a detailed strategy that explains how to build, maintain, replace, and change or improve certain software. The life cycle is a methodology for improving software quality and the development process as a whole.

1.10.1 What is SDLC?

The Software Development Life Cycle, or SDLC, is a method for producing high-quality, low-cost software in the least amount of time. SDLC is a well-structured flow of stages that enables a company to swiftly develop high-quality software that has been thoroughly tested and is ready for production. Within a software organisation, the SDLC is a process that is followed for a software project

The stages of a typical SDLC are depicted graphically in the following diagram.

The steps of a typical Software Development Life Cycle are as follows:

- Requirement analysis

- Planning and requirement gathering

- Software design such as architectural design

- Software development

- Testing

- Deployment and Maintenance

Stage 1: Requirement Analysis

The most critical and fundamental level of the SDLC is requirement analysis. It is carried out by the team's top members, with input from the customer, the sales department, market surveys, and industry domain specialists. This information is then utilised to establish the main project approach and conduct product feasibility studies in the areas of economics, operations, and technology.

Stage 2: Planning and requirement gathering

The planning step also includes determining the project's quality assurance requirements and identifying the project's risks. The goal of the technical feasibility study is to identify the various technical approaches that can be used to successfully implement the project with the least amount of risk.

Following the requirement analysis, the product needs must be properly defined and documented, and they must be approved by the client or market analysts. This is accomplished through the use of an SRS (Software Requirement Specification) document, which contains all of the product requirements that must be designed and developed throughout the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Typically, many design approaches for the product architecture are presented and documented in a DDS - Design Document Specification based on the criteria given in the SRS.

This DDS is reviewed by all essential stakeholders, and the optimal design strategy for the product is chosen based on many parameters such as risk assessment, product robustness, design modularity, budget, and time restrictions.

A design approach identifies all of the product's architectural modules, as well as the product's communication and data flow

representation with external and third-party modules (if any). All of the modules of the proposed architecture's internal design should be thoroughly documented in DDS, down to the tiniest of details.

Stage 4: Software Development

The actual development of the product begins at this stage of the SDLC. During this stage, the programming code is generated according to DDS. Code generation can be done quickly and easily if the design is done in a precise and organised manner.

Developers must adhere to their organization's coding rules, and programming tools such as compilers, interpreters, and debuggers are used to develop code. Code is written in a variety of high-level programming languages, including C, C++, Pascal, Java, and PHP. The programming language is chosen based on the type of software that is being developed.

Stage 5: Testing

As testing activities are mainly included in all phases of SDLC in modern SDLC models, this stage is usually a subset of all stages. This stage, on the other hand, relates to the product's testing stage, during which faults are reported, tracked, corrected, and retested until the product meets the SRS's quality criteria.

Unit, integration, system, and acceptance testing are all performed during this stage.

Stage 6: Deployment and Maintenance

The product is formally released in the appropriate market once it has been thoroughly tested and is ready for deployment. Product deployment can also be done in stages, depending on the company's business strategy. The product might be released in a limited market first, then tested in a real-world setting (UAT- User acceptance testing).

The product may then be released as is or with proposed enhancements in the intended market segment based on the feedback. After a product is launched, it is maintained for existing customers.

1.10.2 SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models. Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Prototyping Model
- Spiral Model
- Incremental Model
- V-Model
- Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Time Boxing Models.

CHECK YOUR PROGRESS

1. What is software?

(A) Software is a documentation and configuration of data

(B) Software is a set of programs

(C) Software is a set of programs, documentation and configuration of data

(D) None of the above

2. SDLC stands for

(A) System Development Life Cycle

(B) Software Design Life Cycle

(C) Software Development Life Cycle

(D) System Design Life Cycle

3. RAD stands for

(A) Relative Application Development

(B) Rapid Application Development

(C) Rapid Application Document

(D) None of the mentioned

4. What is the essence of software engineering?

(A) Requirements Definition, Design Representation, Knowledge Capture and Quality Factors

(B) Maintaining Configurations, Organizing Teams, Channeling Creativity and Planning Resource Use

(C) Time/Space Tradeoffs, Optimizing Process, Minimizing Communication and Problem Decomposition

(D) Managing Complexity, Managing Personnel Resources, Managing Time and Money and Producing Useful Products

5. Which of the following is not a description of planning?

(A) Planning is used to find credible ways to produce results with limited resources and limited schedule flexibility

(B) Planning is finding new personnel resources to support labor intensive development

(C) Planning is identifying and accommodating the unforeseen

(D) Planning is negotiating compromises in completion dates and resource allocation

6. How does a software project manager need to act to minimize the risk of software failure?

(A) Double the project team size

(B) Request a large budget

(C) Form a small software team

(D) Track progress

7. Views of quality software would not include

(A) Optimizing price and performance

(B) Minimizing the execution errors

(C) Conformance to specification

(D) Establishing valid requirement

8. Software measurement is useful to

(A) Indicate quality of the product

(B) Track progress

(C) Assess productivity

(D) All of the above

9. Symptoms of the software crisis would include

(A) Software delivered behind schedule

(B) Software exceeding cost estimate

 (C) Difficult to maintain and Unreliable

(D) All of the above.

10. A systematic approach to software development, as epitomized by the various life-cycle models, is useful in

(A) Helping us understand the nature of the software product

(B) Convincing the customer that we know what we are doing

(C) Filling texts on software engineering

(D) Managing the various activities necessary to get the job done

1.11 SUMMING UP

- A programme or set of programmes containing instructions that offer desired functionality is referred to as software.
- The Characteristics of Software are: Maintainability, Efficiency, Correctness, Reusability, Testability, Reliability, Portability, Adaptability, Interoperability.
- System Software is required to manage computer resources and facilitate application programme execution.
- Software such as PYTHON, MATLAB, AUTOCAD, PSPICE, ORCAD falls under the category of Scientific Software.
- The problem of building meaningful and efficient computer programmes in the time allotted is referred to as a software crisis in computer science.
- Software's complexity and expectations are increasing on a daily basis. Software is becoming more expensive and more complex. This is one of the major causes of Software Crisis.
- Software engineering refers to the application of systematic engineering principles to the creation of software products

and applications. It is a discipline of engineering concerned with the analysis of user requirements, software design, development, testing, and maintenance.

1.12 ANSWERS TO CHECK YOUR PROGRESS

1. Ans. (C)

2. Ans. (C)

3. Ans. (B)

4. Ans. (D)

5. Ans. (D)

6. Ans. (D)

7. Ans. (D)

Reason: minimizing the execution errors would not included in views of quality software.

8. Ans. (D)

9. Ans. (D)

Reason: When software delivered behind schedule, software exceeding cost estimate, unreliable and difficult to maintain then it is said that software crisis.

10. Ans. (D)

Reason: A systematic approach to software development, as epitomized by the various life-cycle models, is useful in managing the various activities necessary to get the job done.

1.13 POSSIBLE QUESTIONS

1. What are the important categories of software?

2. Describe the software development process in brief.

3. Name two tools which are used for keeping track of software requirements?

4. What is feasibility study?

5. What are functional and non-functional requirements?

1.14 REFERENCES AND SUGGETSED READINGS

- https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/

- https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

- Fundamentals of Software Engineering, Rajib Mall

UNIT 2: SOFTWARE PROCESS MODEL

Unit Structure:

2.1 INTRODUCTION

A software process (also known as software methodology) is a collection of related operations that leads to software production. These actions could include creating new software or changing an existing one.

The following four activities are required in any software development process:

- Software specification (or requirements engineering): Define the software's main functions and constraints.

- Software design and implementation: It will be necessary to design and programmed the software.

- Software verification and validation: The software must meet the customer's requirements and correspond to its specifications.

- Software evolution (software maintenance): The software is constantly updated to meet changing customer and market needs.

2.2   UNIT OBJECTIVES

A Software Process Model gives a roadmap for software engineering work. It defines the flow of all activities, actions and tasks. The main Objective is to introduce the generic concept of software engineering process model with the concept of software process and software process models. Five traditional process models have been discussed with their pros and cons in this chapter. Verification and Validation also discussed for specifications and standards because software system meets the need.

2.3   SOFTWARE PROCESS MODEL

A software process model is a detailed description of a software process from a certain point of view. A software process model is an abstraction of the real process that is being represented, as models are by their very nature simplifications. Activities that are part of the software process, software products, and the roles of persons involved in software engineering may be included in process models.

Types of Software Process Model

There are a variety of process models available to satisfy various requirements. Software development life cycle (SDLC) models are one of the most fundamental parts of the software development process. There are a variety of software development life cycle models designed to achieve certain goals. These models are defined at different stages of the process and development module in which they are implemented. The following are the most commonly used, popular, and important SDLC models:

1.3.1 Waterfall Model

The first Process Model to be introduced was the Waterfall Model. It is also known as a linear-sequential life cycle model. It is very simple to understand and use. In this waterfall model, each phase must be completed before moving on to the next, and the phases do not overlap. The waterfall Model illustrates the software development process in a linear sequential flow.

The Waterfall model is the earliest SDLC approach that was used for software development.

Design:

The Waterfall Approach was the first SDLC Model to be widely utilized in Software Engineering to ensure project success. The entire software development process is separated into several phases using "The Waterfall" approach. Typically, the output of one phase acts as the input for the following phases in this Waterfall approach. The different phases of the Waterfall Model are depicted in the following diagram.
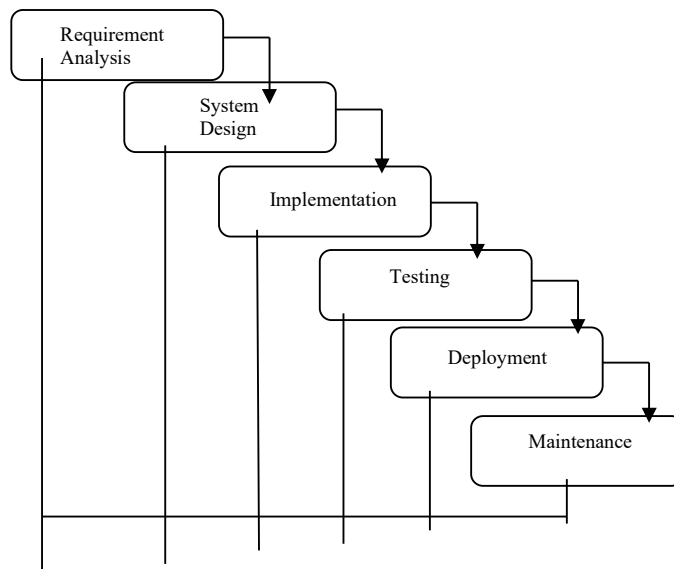
Fig 1: Waterfall Model

The sequential phases in Waterfall model are −

- Requirement Gathering and analysis − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document. Business analysts gather requirements, which are then analyzed by the team. Based on their discussions with the client, business analysts will document the requirements.

- System Design − this phase examines the requirements specifications from the previous phase and prepares the system design. This system design helps in designing the overall system architecture as well as describing hardware and system requirements.

- Implementation − The system is first built as discrete programs called units, which are then merged in the next phase, using inputs from the system design. Unit testing is the process of developing and testing each unit for its functioning.

- Integration and Testing − After each unit has been tested, all of the units built during the implementation phase are integrated into a system. The entire system is then tested for any flaws or failures after it has been integrated.

- Deployment of system − The product is deployed in the client environment or released into the market once functional and non-functional testing is completed.

- Maintenance − In the client environment, there are a few challenges that arise. Patches are released to remedy the issues. In order to improve the product, newer versions have been produced. Maintenance is carried out in order to bring about these changes in the customer's environment.

Advantages:

Waterfall development has the advantage of allowing for departmentalization and control. A schedule can be created with deadlines for each step of development, and a product can be guided through the various phases of the development process one by one.

The following are some of the key benefits of the Waterfall Model:

- It's simple to comprehend and utilize.

- Because of the model's rigidity, it's simple to manage. There are specified deliverables and a review mechanism for each phase.

- One phase at a time is processed and completed.

- For smaller projects with well-defined needs, this method works effectively.

- Stages that are well defined.

- Milestones that are well understood

- Tasks are simple to organize.

- Both the process and the outcomes are well documented.

Disadvantages:

Waterfall development has the problem of not allowing for much reflection or correction. It's quite tough to go back and fix something that wasn't well-documented or considered in the design stage once an application has reached the testing stage.

The following are the key drawbacks of the Waterfall Model:

- Until late in the life cycle, no working software is developed.

- There is a lot of risk and uncertainty.

- For sophisticated and object-oriented projects, this is not a good model.

- For long-term projects, this paradigm is inadequate.

- Not appropriate for projects with a moderate to high risk of change in requirements. As a result, this process model has a high level of risk and uncertainty.

- Within stages, it's tough to assess development.

- Changes in requirements cannot be accommodated.

- Changing the scope of a project during its life cycle might lead to its termination.

- Integration is done as a "big-bang" towards the end, which prevents any technological, or business bottlenecks or issues from being identified early.

1.3.2 Prototyping Model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. The prototyping model can be considered to be an extension of the waterfall model. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. Prototyping Model is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved. It also creates base to produce the final system or software. It works best in scenarios where the project's requirements are not known in detail. It is an

iterative, trial and error method which takes place between developer and client. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

It has limited functional capabilities, low reliability, or inefficient performance as compared to the actual software. A prototype can be built very quickly by using several shortcuts. The shortcuts usually involve developing inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up rather than by performing the actual computations. Normally the term rapid prototyping is used when software tools are used for prototype construction. For example, tools based on fourth generation languages (4GL) may be used to construct the prototype for the GUI parts.

Necessity of the prototyping model

The prototyping model is advantageous to use for specific types of projects. In the following, we identify three types of projects for which the prototyping model can be followed to advantage: It is advantageous to use the prototyping model for development of the graphical user interface (GUI) part of an application. Through the use of a prototype, it becomes easier to illustrate the input data formats, messages, reports, and the interactive dialogs to the customer. This is a valuable mechanism for gaining better understanding of the customers' needs. In this regard, the prototype model turns out to be especially useful in developing the graphical user interface (GUI) part of a system. For the user, it becomes much easier to form an opinion regarding what would be more suitable by experimenting with a working user interface, rather than trying to imagine the working of a hypothetical user interface.

The GUI part of a software system is almost always developed using the prototyping model. The prototyping model is especially useful when the exact technical solutions are unclear to the development team. A prototype can help them to critically examine the technical issues associated with product development. For example, consider a situation where the development team has to write a command language interpreter as part of a graphical user interface development. Suppose none of the team members has ever written a

compiler before. Then, this lack of familiarity with a required development technology is a technical risk. This risk can be resolved by developing a prototype compiler for a very small language to understand the issues associated with writing a compiler for a command language. Once they feel confident in writing compiler for the small language, they can use this knowledge to develop the compiler for the command language. Often, major design decisions depend on issues such as the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype is often the best way to resolve the technical issues.

An important reason for developing a prototype is that it is impossible to "get it right" the first time. As advocated by Brooks [1975], one must plan to throw away the software in order to develop a good software later. Thus, the prototyping model can be deployed when development of highly optimized and efficient software is required. From the above discussions, we can conclude the following:

The prototyping model is considered to be useful for the development of not only the GUI parts of a software, but also for a software project for which certain technical issues are not clear to the development team.

Software is built in two ways, as depicted in Figure: prototype construction and iterative waterfall-based software development.

Prototype construction: The development of prototyping begins with the gathering of basic requirements. A prototype is produced after a quick design is completed. The customer is asked to evaluate the prototype that has been created. Based on the customer feedback, the requirements are refined and the prototype is suitably modified. This cycle of collecting consumer feedback and making changes to the prototype continues until the customer approves it.

Iterative development: The actual software is produced utilizing an iterative waterfall approach after the customer approves the prototype. Regardless of whether or not a functioning prototype is available, the SRS document must be created because it is essential for later phases such as traceability analysis, verification, and test case creation.

However, for GUI parts, the requirements analysis and specification phase becomes redundant since the working prototype that has been approved by the customer serves as an animated requirements

specification. The prototype's code is usually discarded. The expertise gained by constructing the prototype, on the other hand, is invaluable when it comes to developing the genuine system.

Despite the fact that building a throwaway prototype incurs more costs, the overall development cost for systems with unclear client requirements and systems with unsolved technical challenges is frequently lower than for an analogous system produced using the iterative waterfall model. Many customer requirements are adequately defined and technological concerns are overcome by experimenting with the prototype after it is built and submitted for user evaluation. This reduces the number of client requests for changes in the future, as well as the accompanying redesign expenses.

Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

Step 1: Requirements gathering and analysis

Requirement analysis is the first step in a prototype model. The system's requirements are outlined in depth at this phase. The system's users are interviewed as part of the process to learn what they expect from it.

Step 2: Quick design

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

Step 3: Build a Prototype

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.
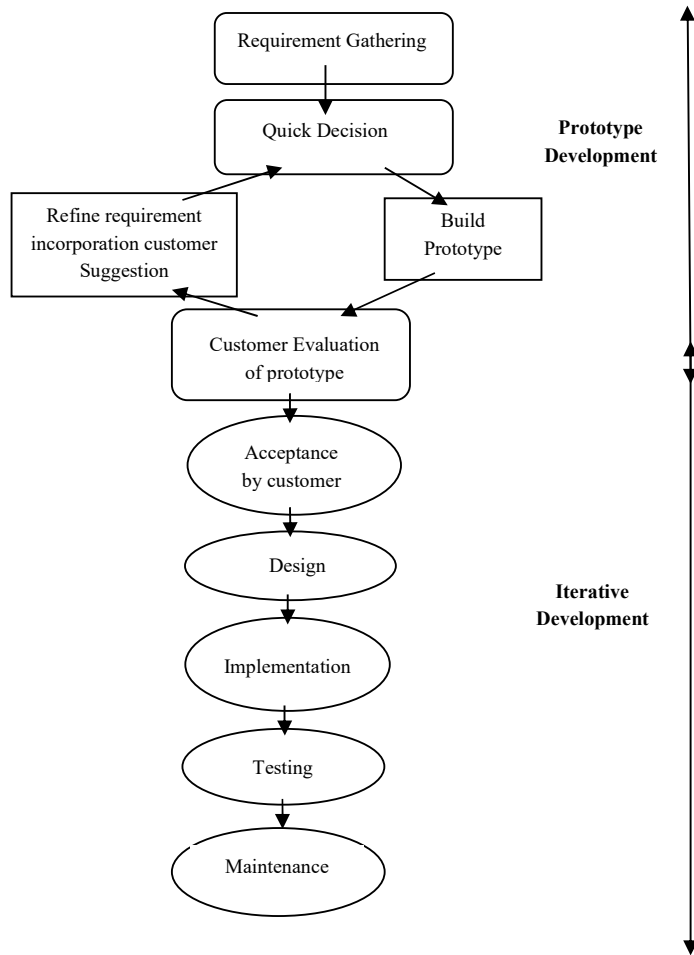
Figure 2: Prototyping model of software development.

Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes

routine maintenance for minimizing downtime and prevents large-scale failures.

Advantages of the Prototyping Model

This model is the most appropriate for projects that suffer from technical

and requirements risks. These risks can be mitigated with a well-built prototype. Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process. Prototyping is also considered a risk reduction activity, so missing functionality can be identified, lowering the risk of failure. Customers are satisfied because they can feel the product at an early stage. There will be very little probability of rejection of the software. Improved software development solutions are made possible by faster user feedback. Allows the client to compare if the software code matches the software specification. It helps you to find out the missing functionality in the system. It also indicates the functions that are complicated or challenging. Because it is a simple model, it is simple to comprehend. Building the model does not necessitate the use of professional experts. The prototype aids in the understanding of the customer's requirements. Changes and even discarding prototypes are possible. Future users of the software system may benefit from early training provided by prototypes.

Disadvantages of the Prototyping Model

The prototype model can increase the cost of development for projects that are routine development work and do not suffer from any significant risks. Even when a project is susceptible to risks, the prototyping model is effective only for those projects for which the risks can be identified upfront before the development starts. Since the prototype is constructed only at the start of the project, the prototyping model is ineffective for risks identified later during the development cycle. The prototyping model would not be appropriate for projects for which the risks can only be identified after the development is underway. Prototyping is a time-consuming and slow procedure. The cost of building a prototype is completely wasted because the prototype is eventually discarded. Excessive modification requests may be encouraged through prototyping. Customers may not be willing to participate in an iteration cycle for an extended period of time. When the customer evaluates the prototype each time, there may be much too many differences in

software needs. Because the needs of the clients are always changing, there is a lack of documentation. It is quite tough for software engineers to meet all of the client requests. When a client is unhappy with the initial prototype, he or she may lose interest in the ultimate product.

1.3.3 Spiral Model

The spiral model is one of the most prominent Software Development Life Cycle models for risk management. It resembles a spiral with several loops in diagrammatic depiction. The spiral's exact number of loops is unclear, and it varies from project to project. A Phase of the software development process is defined as each loop of the spiral. Depending on the project risks, the project manager might change the number of phases required to build the product. The project manager plays an important role in developing a product utilizing the spiral model since the number of phases is dynamically determined by the project manager.

The spiral's radius at any given moment symbolizes the project's expenses (cost), while the angular dimension shows the current phase's progress.

The phases of the Spiral Model are depicted in the diagram below: –
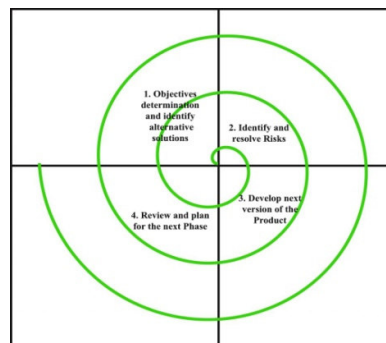
Fig 3: Spiral Model

As illustrated in the diagram above, each phase of the Spiral Model is divided into four quadrants. The following sections go through the functions of these four quadrants:

1. Objectives determination and identify alternative solutions: At the outset of each step, customers' requirements are gathered,

and objectives are identified, elaborated, and analyzed. Then, in this quadrant, alternative solutions for the phase are given.

2. Identify and resolve Risks: All viable solutions are reviewed in the second quadrant in order to choose the best one. The risks connected with that solution are then identified, and the risks are mitigated using the best technique possible. The Prototype is constructed at the end of this quadrant for the best possible solution.

3. Develop next version of the Product: During the third quadrant, the identified features are developed and verified through testing. The next edition of the software is available at the end of the third quadrant.

4. Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so far developed version of the software. Finally, the planning for the following phases begins.

Risk Handling in Spiral Model

A risk is anything that could prevent a software project from being completed successfully. The spiral model's most essential aspect is how it handles unforeseen hazards once the project has begun. The development of a prototype makes such risk resolutions easier. The spiral approach supports risky copying by allowing for the creation of a prototype at each stage of software development.

Risk management is also supported by the Prototyping Model, however risks must be fully identified prior to the commencement of the project's development activity. However, in real life, project risk may arise after development work has begun; in this scenario, the Prototyping Model cannot be used. The product's features are evaluated and examined in each phase of the Spiral Model, and the risks at that moment in time are identified and resolved through prototyping. As a result, this paradigm is far more adaptable than other SDLC models.

Why Spiral Model is called Meta Model?

Because it encompasses all other SDLC models, the Spiral model is referred to as a Meta-Model. The Iterative Waterfall Model, for example, is represented by a single loop spiral. The Classical Waterfall Model's progressive approach is included into the spiral model. The spiral model employs the Prototyping Model's risk-handling technique of developing a prototype at the start of each

phase. The spiral model can also be thought of as a support for the evolutionary model, with iterations along the spiral serving as evolutionary layers upon which the entire system is created.

Advantages of Spiral Model:

The Spiral Model has a number of advantages.

1. Risk Handling: Due to the risk analysis and risk management at each phase, the Spiral Model is the best development model to follow for projects with many unknown risks that arise as the development progresses.

2. Good for large projects: In large and complex undertakings, the Spiral Model is recommended.

3. Flexibility in Requirements: Using this paradigm, change requests in the Requirements at a later stage can be accurately implemented.

4. Customer Satisfaction: Customers can observe the product's progress throughout the early stages of software development, and so become familiar with the system by using it before the final product is completed.

Disadvantages of Spiral Model:

The spiral model has several major drawbacks, which are listed below.

1. Complex: Other SDLC models are substantially more sophisticated than the Spiral Model.

2. Expensive: The spiral model is not appropriate for small projects due to its high cost.

3. Too much dependability on Risk Analysis: Risk Analysis plays a critical role in the project's success. The development of a project employing this strategy will be a failure without a large number of highly experienced professionals.

4. Difficulty in time management: Time estimation is challenging because the number of phases is unknown at the beginning of the project.

1.3.4 Incremental Model

This life cycle approach is also known as the incremental or successive version model. In this life cycle approach, the customer is first given a simple working system with only a few basic features. Iteratively, until the required system is accomplished, successive versions are implemented and supplied to the customer. Figure 4 illustrates the incremental development model.
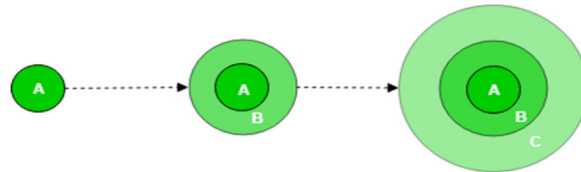
Fig 4: Incremental Model

A, B, C are modules of Software Product that are incrementally developed and delivered.

The software requirements are initially broken down into multiple modules or features, which can then be created and delivered progressively under the incremental life cycle model. At any given time, only the next increment's plans are made, with no long-term planning. As a result, accommodating client requests for changes becomes less difficult. The system's core features are developed first by the development team. The core or fundamental features are those that do not require the use of any other features' services. Non-core features, on the other hand, require services from core features. Following the development of the first basic features, further functionalities are added in succeeding editions to refine them into higher degrees of capability. The iterative waterfall model is typically used to produce each incremental version.

Customer feedback on the delivered version is acquired when each succeeding version of the software is built and delivered to the customer, and these feedbacks are included into the following version. Each version of the software that is supplied to the customer adds new features and refines those that have already been delivered. Figure 5 depicts the incremental model in schematic form. After the requirements gathering and specification, the requirements are split into several versions. The next version is built using an iterative waterfall approach of development and deployed at the customer site, starting with the core (version 1). The whole software is

deployed after the last (marked as version n) has been built and deployed at the client site.
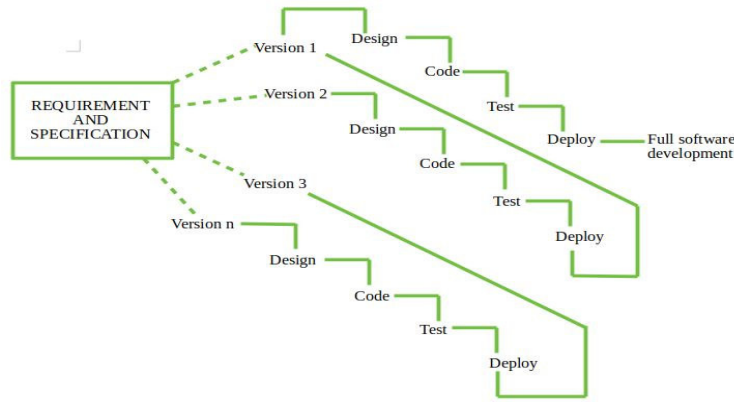
Figure 5: Incremental model of software development.

Advantages:

There are a number of benefits to the incremental development paradigm. The following are two prominent examples:

Error reduction: Because the customer uses the core modules from the beginning, they are thoroughly tested. This minimizes the likelihood of faults in the final product's core modules, resulting in increased software reliability.

Incremental resource deployment: This paradigm eliminates the requirement for the customer to commit significant resources to the system's development all at once. It also saves the growing organization from having to deploy enormous amounts of resources and staff all at once for a project.

Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly). Uses divide and conquer for breakdown of tasks. Lowers initial delivery cost. Incremental Resource Deployment.

Disadvantages:

Requires good planning and design. Total cost is not lower. Well defined module interfaces are required.

1.3.5 Time Boxing Model

As with the iterative enhancement approach, development is done iteratively in the time boxing model. In the time boxing approach, however, each cycle is completed within a predetermined timeframe. The functionality that has to be built is scaled down to match the timeframe. Furthermore, each timebox is divided into a set of predefined phases, each of which completes a distinct task (analysis, implementation, and deployment) that can be completed independently. This approach also requires that each stage's time duration be roughly equal, so that the pipelining concept can be used to save development time and product releases. There is a dedicated team for each stage so that the work can be done in pipelining. As a result, stages should be set so that each step completes a logical unit of work that serves as the input for the following stage.

In addition to the benefits of the iterative model, the time boxing concept has several additional benefits. The time boxing model has a number of advantages and problems.
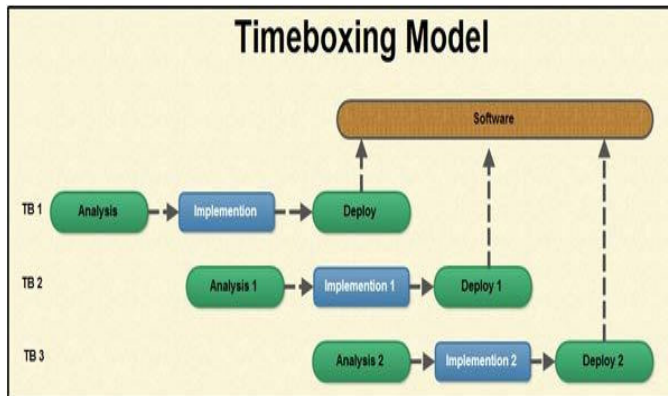

Figure 6: Time boxing model.

The Advantages and Disadvantages of the Time boxing Model

Advantages:

Speeds up the development process and shortens the delivery time. Ideally suited to developing projects with a variety of features in a short amount of time.

Disadvantages:

Project management is becoming more difficult. Not recommended for projects where the entire development process cannot be broken down into many iterations of almost equal time.

1.4 VERIFICATION AND VALIDATION

Verification and Validation is the process of determining whether or not a software system meets the needed specifications and standards. Verification and validation, according to Barry Boehm, are:

Verification: Are we building the product right?

Validation: Are we building the right product?

Validation comes after Verification.

Verification	→	Validation
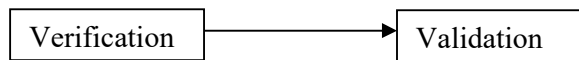
1.4.1 Verification

Verification is the process of ensuring that software meets its objectives without errors. It's the process for determining whether the product being developed is correct or not. It determines whether the developed product meets our requirements. Verification is Static Testing.

Activities involved in verification:

- Inspections
- Reviews
- Walkthroughs
- Desk-checking

1.4.2 Validation

Validation is the process of determining whether a software product meets the required standards, or in other words, whether it meets the product's high level requirements. It is the process of verifying product validation, or ensuring that the product we are building is correct. It is a validation between the actual and desired product. Validation is the Dynamic Testing.

Activities involved in validation:

- Black box testing

- White box testing

- Unit testing

- Integration testing

Difference between verification and validation testing

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as static testing.	Validation is also known as dynamic testing.
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like functional testing, system testing, integration, and User acceptance testing.
It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
Quality assurance comes under verification testing.	Quality control comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the	After verification testing,

validation testing.	validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or not.	In this type of testing, we can validate that the user accepts the product or not.

CHECK YOUR PROGRESS

Q.1. In which model the Project risk factor is considered?

(A) Spiral model.

(B) Waterfall model.

(C) Prototyping model

(D) Incremental model

Q.2. What is the most important use of the incremental model?

(A) Customers can respond to each increment

(B) Easier to test and debug

(C) To use it when we need to get a product to the market early

(D) Easier to test and debug & use it when we need to get a product to the market early

Q.3. The spiral model was the first time proposed by

(A) IBM

(B) Pressman

(C) Barry Boehm

(D) Royce

Q.4. What is the disadvantage of the Spiral Model.

(A) Does n't work well for smaller projects

(B) The high amount of risk analysis

(C) Additional Functionality can be added later

(D) Strong approval and documentation control

Q.5. Where is the prototyping model of software development well suited?

(A) When requirements are well defined.

(B) For projects with large development teams.

(C) When a customer cannot define requirements clearly.

(D) None of the above.

Q.6. Design phase is followed by _____ .

(A) Coding

(B) Testing

(C) Maintenance

(D) None of the above.

Q.7 Which of the following activities of the generic process framework delivers a feedback report?

(A) Deployment

(B) Planning

(C) Modeling

(D) Construction

Q.8 Which one of the following activities is not recommended for software processes in software engineering?

(A) Software Evolution

(B) Software Verification

(C) Software Testing & Validation

(D) Software designing

Q.9 The _____ and _____ are the two major dimensions encompassed in the Spiral model.

(A) Diagonal, Perpendicular

(B) Perpendicular, Radial

(C) Angular, diagonal

(D) Radial, Angular

Q.10 Which parameters are essentially used while computing the software development cost?

(A) Hardware and Software Costs

(B) Effort Costs

(C) Travel and Training Costs

(D) All of the above

1.5 SUMMING UP

- A Software Process Model gives a roadmap for software engineering work. It defines the flow of all activities, actions and tasks.
- A software process model is an abstraction of the real process that is being represented, as models are by their very nature simplifications. Activities that are part of the software process, software products, and the roles of persons involved in software engineering may be included in process models.
- The following are the most commonly used, popular, and important SDLC models:
  - Waterfall Model
  - Prototyping Model
  - Spiral Model
  - Incremental Model
  - Time Boxing Model
- Verification is the process of ensuring that software meets its objectives without errors. It's the process for determining whether the product being developed is correct or not.
- Validation is the process of determining whether a software product meets the required standards, or in other words, whether it meets the product's high level requirements. It is the process of verifying product validation, or ensuring that the product we are building is correct.

1.6 ANSWERS TO CHECK YOUR PROGRESS

1. Ans. A

2. Ans. D

3. Ans. C

4. Ans. A

5. Ans. C

7. Answer: Deployment

Explanation: The deployment phase is the last phase of the software development life cycle in which the software product is delivered to its end-user, who further assesses its performance and revert back

with the feedback if anything is required or missing as per the formulated evaluation.

8. Answer: b) Software Verification

Explanation: Software verification is mainly considered for implementing and testing activities.

9. Answer: D) Radial, Angular

Explanation: The cumulative cost is represented by the radial dimension, whereas the angular dimension represents the progress made in the completion of each consecutive cycle. Each loop in the spiral model depicts the phase.

10. Answer: D) All of the above

Explanation: Estimation cost works out on assessing the amount of effort required to complete each activity, followed by calculating the total cost of activities.

1.7 POSSIBLE QUESTIONS

1. Why document a development process?

2. What is a data structure-oriented software design methodology? How is it different from the data flow-oriented design methodology?

3. What is SDLC? What are different SDLC models?

4. Describe classical waterfall model and iterative development model of Software development. Draw appropriate diagrams. Compare the two models.

5. Discuss about prototyping model. Explain its merits and Demerits.

6. Discuss in detail about Time Boxing model.

7. Describe how incremental process models are better than water fall model.

8. What process models will you use in various projects?

9. What are perspective process models?

1.8 REFERENECS AND SUGGESTED READINGS

- https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/

- https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

- Fundamentals of Software Engineering, Rajib Mall

UNIT 3: SOFTWARE REQUIREMENTS AND ANALYSIS

Unit Structure:

3.1 INTRODUCTION

The Software requirements and analysis phase starts after the feasibility study phase gets completed and the project is found to be feasible and technically sound. The primary objective of the software requirements analysis and specification phase is to have a clear understanding of the customer's requirements and to organize these requirements systematically in a specification document. This phase consists of the following two activities:

• Requirements Gathering and Analysis

• Requirements Specification

Identifying the user requirements properly is quite a tedious job. It combines the processes of describing, analyzing, documenting and validating the services, requirements and constraints related to the software. All these processes, in combination, are called Software Requirement engineering or simply, software requirement.

STOP TO CONSIDER
Requirements may serve a dual function: • As the basis of a bid for a contract • As the basis for the contract itself

Space for learners:

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Be familiar with software requirements

- Understand the concepts and significance of Software Requirement Analysis

- Differentiate between the types of Software Requirements

- Understand the structure of Software Requirement Specification

- Characteristics of SRS

3.3 SOFTWARE REQUIREMENT

The software requirements include a precise description of the functionalities and features of the target software system. These requirements specify what the users would expect from the software product. The software requirements can be unknown or known, expected or unexpected and hidden or obvious from the users' point of view. The objective of software requirement is to specify the requirements of the software product in a consistent, unambiguous and concise manner- with the help of appropriate formal notations. The requirement specification would focus on the 'What" part of the software rather than on "how". It starts with the requirements gathering process. The System analyst starts requirements gathering activity by collecting all information from the customer which could be used to develop the requirements of the system. Then, he analyses the information collected to get a thorough understanding of the system to be developed. The objective is to get rid of all ambiguities and inconsistencies from the initial problem perception by the customer

3.3.1 Software Requirement Statement

A software requirement statement is a document that mentions the intended use, challenges and features of the software application.

Software system requirements can be broadly classified into three groups-

> **User Requirement Document**- It contains natural language statements related to the services provided by the system, along with the operational constraints. It also includes diagrammatic representations related to the services offered. Such a document is usually written for the customers.

> **System Requirement Document**- It is s structured document that provides detailed descriptions of the services. This document is usually written as a contract between a contractor and a client.

> **Software Specification Document**- It contains a detailed description of the software that serves as a foundation for the implementation or design of the software. Such a document is usually written for the developers.

STOP TO CONSIDER

The software requirements include a precise description of the functionalities and features of the target software system.

3.4 TYPES OF REQUIREMENTS

There may be three types of software requirements, namely

a) Functional requirements, and

b) Nonfunctional requirements.

c) Domain requirements

3.4.1 Functional Requirements

The functional requirements of a software system details the services or functionalities that the user expects it to provide. These requirements will describe how the software system can react to a specific set of inputs and how it should ideally behave in a particular situation.

The functional requirements focus on the functionalities required by the

users from the software system. The users or clients expect the software to perform a specific set of functions.

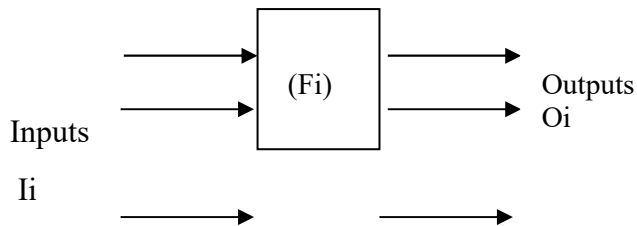The functional view of a software system is shown in the figure below-

Fig:3.1

Each function (Fi) of the software system can be thought of as a conversion from a specific set of input data (Ii) to the corresponding set of output data (Oi). From the user's perspective, he should be able to accomplish some meaningful piece of work with the help of the function.

3.4.2 Non-Functional Requirements

Constraints are part and parcel of any software system. The functionalities and services offered by the system come with many constraints. These include constraints related to the process of development, timing constraints, constraints related to standards etc. The non-finctional requirements define the constraints associated with a software system. They are not exactly concerned about a particular function delivered by the software system. Rather, they may be related to system properties like response time, storage and reliability. They may also define the constraints related to the system such as data representation used in the interfaces and capabilities of the I/O devices.

To be precise, nonfunctional requirements are concerned with the system characteristics that the functions cannot express. These include - portability, maintainability , usability and so on. It may also include accuracy of results, reliability issues, implementation issues and issues related to human - computer interface etc.

These requirements primarily deal with issues such as:
- Portability issues

- Usability issues
- Maintainability issues
- Security issues
- Scalability issues
- Reliability issues
- Reusability issues
- Flexibility issues
- Performance issues

The differences between Functional and Non-functional requirements are shown below in a tabular form

Functional Requirements	Non-Functional Requirements
Defines a software system or its components.	Defines the quality attributes related to a software system.
Specifies what the software system should do	Describes constraints on how the software should fulfill its functional requirements
These requirements are specified by the user.	These requirements are specified by technical people such as software developers, architects, technical leaders and so on
It is a mandatory requirement	It is not a mandatory requirement
Captured with regards to use case.	Captured as a quality attribute.
It is applied at the component level.	It is applied to a system as a whole.
Allows to check the functionality of the software.	Allows to ascertain the software performance
Carries out functional testing such as integration, System, API Testing, End to End etc	Performs Non-Functional Testing such as Usability, Performance, Security testing, Stress etc
Easy to define.	Comparatively difficult to define.

3.4.3 Domain Requirements

These requirements describe system features and characteristics that specify the domain. These may include constraints on prevailing requirements , new functional requirements or may even define specific computations. The system may not work properly if the domain requirements are not fully satisfied.

3.5   REQUIREMENTS ENGINEERING PROCESS

The requirement engineering process refers to the procedure of gathering specific software requirements from the concerned client, analyzing and documenting them for future reference.

The objective of this process is to prepare and maintain a descriptive and comprehensive SRS (System Requirements Specification) document.

The entire requirement engineering process consists of four steps. These are –

- Feasibility Study

- Requirement Gathering

- Software Requirement Specification

- Software Requirement Validation

3.5.1 Feasibility Study

When a client assigns an organization the task of developing the desired software product, he often provides a rough idea about the expected features from the software or the functions it must perform. Taking a reference from this raw information, the analysts of the organization makes a complete study about whether it is feasible to develop the software system, with the desired functions embedded in it. Such a study can be termed as feasibility study. The output of this study should be a feasibility study report with adequate recommendations and comments for the management for them to consider whether or not the project should be undertaken.

We will discuss it in detail in the next section

3.5.2 Requirement Gathering

If the feasibility study gives a positive report about undertaking the project, the next phase starts, that is, collecting requirements from the client or user. Analysts and technical staffs of the organisation communicate with the end-users/client to understand their ideas. Thus, they will try to apprehend what the software must provide and the features they would like the software to have. Requirements gathering itself is an art. The person who is entrusted with the responsibility of gathering requirements should have the knowledge of when to gather and what information to gather and by using what resources. The requirements regarding organisation, which include information related to its policies, objectives, organisational structure and user staff are gathered. Moreover, it includes information regarding job function, information about work schedules, work flow and working procedure. Below, we are going to discuss some of the requirements gathering tools.

1. **Record review**: A review of all the recorded documents of the organization such as procedures, manuals, forms and books, are reviewed to understand the format and

functions of the present system. This technique consumes more time.

2**. On-site observation**: In case there exists such a system, the actual

site of the system is visited

to get a close view of the system. It enables the analyst to detect the problems of the existing

system.

3. **Interview**: The system analyst interacts with the staff to identify their

requirements. It requires experience in arranging the interview, avoiding arguments and evaluating the outcome.

4. **Questionnaire**: It is an effective means to acquire customer requirements with less effort. This produces a written document about the requirements. It considers the responses of a large number of participants at the same time and examines answers to the queries.

3.5.3 Software Requirement Specification

SRS is a document created by a system analyst. After the user requirements are gathered from different stakeholders, the system analysts of the organization create a document called the SRS (Software Requirement Specification) document. The SRS will define how the proposed software would interact with external interfaces, hardware, system response time, operational speed, portability across platforms, recovery after crashing, security, maintainability, quality, constraints and so on.

Please note that the user requirements are usually received in a natural language format. It is the job of the system analyst to write all these requirements in a technical language in the SRS document. He should be able to write in a language so that it can be useful for the development team.

3.5.4 Software Requirement Validation

After the SRS is developed as per user specifications, it is again validated with the requirements of the user. It may so happen that the user may ask for some   illegal solutions. The experts may also interpret the user requirements incorrectly. Such issues, if not addressed at the very beginning, may lead to an exponential increase in cost. Therefore, it is important to check the requirements against the below-mentioned conditions -

- If it is practical to implement requirements
- If the requirements are valid and conform to the domain and functionality of the software
- If the requirements are complete
- If the requirements can be demonstrated
- If any ambiguities exist

STOP TO CONSIDER

The process of finding out, analyzing, documenting and checking these services and constraints is called Requirements Engineering.

3.6 FEASIBILITY STUDY

3.6.1 What is Feasibility Study?

The objective of this study is to establish the reasons for developing the software, in a way to make it conform to agreed standards, acceptable to users and flexible to change.

The feasibility study is aligned with the goal of the organization. It analyzes if it is practicable to materialize the software in terms of cost constraints, implementation, project contribution to the organization and the perceived objectives and values of the organization. It mainly looks into the technical aspects of the project, which include productivity, integration ability, maintainability and usability.

The output of this whole exercise should be in the form of a feasibility study report. It should include sufficient comments and recommendations so as to help the management decide in favour of or against undertaking the particular project. The business strategists in the organization analyse the feasibility study report to find out if the software can fulfill the user requirements and the software can generate profit at the same time. If the project is found to be feasible technologically, practically and financially- it will be given a go-ahead. Else, it will be discarded.

3.6.2 Feasibility Study: Why it is required

The feasibility study is an extremely important stage of the Software Project Management Process. After the feasibility study gets completed, the organisaton gets a clear idea of whether it is profitable or practically feasible to continue with the proposed project. The feasibility study also helps the organization to identify the risk factors associated with developing and deploying the system. Based on the findings of the feasibility study, the organization can plan for the risk analysis and zero in on the business alternatives. By analyzing various project parameters mentioned in the feasibility study report, the organization can also enhance the success rate of the proposed software.

3.6.3 Types of Feasibility Study

Three different types of feasibility study may exist, namely

1. **Technical Feasibility** – It evaluates the latest technologies that are required to fulfill customer requirements, conforming to the specified time and budget. It also takes into account the resources at hand (software, hardware etc). The feasibility study will report if the required technologies and technical resources are available, adequately enough, for developing the software. In addition, the feasibility study analyses the capabilities and skills of the technical team. It assesses if existing technology can be employed, if any technology upgradation is required and the cost of

deployment/maintenance of the upgraded technology. Technical feasibility also performs the following tasks-

- • Determines whether relevant technology is established and stable.

- • Analyzes the technical capabilities of the software development team.

- • Ascertains that the technology picked for software development has a huge number of users, so that in case any problem arises or improvements are required, they can be consulted.

2. **Operational Feasibility** – This is a type of feasibility that determines the range within which the software performs a designated series of levels in order to fulfill customer requirements and resolve business issues. Here, the level of providing service as per user requirements is analysed. It also studies how easy or tough it will be to operate and maintain the software after deployment. Other operational scopes such as usability of the product, acceptability of the solutions provided by the development team, etc are also studied. Operational feasibility carries out the following tasks:

- • Determines whether the problems in user requirements can be regarded as of high priority.

- • Determines whether the suggested solution forwarded by the software development team is acceptable.

- • Ascertains whether users are ready to adapt to new software.

- • Finds about whether the organization is satisfied by the software development team's proposed alternative solutions

3. Economic Feasibility – This type of feasibility determines whether the software will be able to produce financial profits for an organization. It makes a cost-benefit analysis of the software development project. A detailed analysis is carried out to assess the total cost of the project, including the cost of design, development, procurement of resources, operational cost and others. Then, it will be

analysed if developing the project will be profitable financially or not. The Software is said to be economically feasible if it highlights the issues listed below.

• Cost incurred on the development of the software will produce long-term gains for an organization.

• Cost required to carry out full software investigation such as requirements elicitation and requirements analysis.

• Cost of software, hardware, development team, and training.

3.6.4 The Feasibility Study Process

The entire process of a feasibility study involves the following steps -

- ➢ Information assessment
- ➢ Collection of Information
- ➢ Report writing
- ➢ General information

Let us briefly study the steps involved in a feasibility study.

• **Information assessment:** Identifies information regarding whether the system aid in achieving the goals of the organization. Apart from this, it verifies whether that the system can be implemented by making use of new technology and within the budget. It also verifies whether the system to be developped can be integrated with the existing system.

• **Information collection:** It states the sources from where information regarding the software can be acquired. In general, the users (who will be operating the software), the organisation where the software will be used, and the software development team (which has a clear understanding of user requirements and knows how to accomplish them in software) act as the main source of information.

• **Report writing:** The software development team uses a feasibility report, to mark the conclusion of the feasibility study. The report includes recommendations on whether the development of the software should be continued. Information about changes regarding software scope, schedule, and budget and suggestions with respect to any requirements in the system is also included in the report .

• **General information:** This describes the purpose along with the scope of the feasibility study. It also elucidates system overview, acronyms and abbreviations, project references, and points of contact to be used. **Project references** give a list of references used for the preparation of the document**. System overview** gives a description of the organization responsible for the software development, system category, system name or title, operational status, and so on. **Acronyms and abbreviations** describe a list of the terms used in this document accompanied by their meanings. **Points of contact** provide a list of organizational contact points with the users for future information and coordination. For example, users might require assistance for solving a particular problem (such as troubleshooting)

3.6.5 Outcome of The Feasibility Study

Based on the information assessed (about the requirements) , information collected and report written, the following list of questions is obtained-

a) What will happen if the system is not implemented?

b) What are the present problems related to the project?

c) How will the proposed software system help?

d) What will be challenges related to integration?

e) Is there any requirement for new technology?

f) Is there any requirement for new skill sets and team members?

g) What facilities would the proposed system support?

3.7 ELICITATION OF REQUIREMENTS

In the process of requirement elicitation, the requirements of the proposed software are ascertained by communicating with end-users, system users , clients and other stakeholders.

3.7.1 The Process of Requirement Elicitation

The process of requirement elicitation can be described with the help of the following diagram:

Fig 3.2 Process of requirement elicitation

- **Requirements gathering –** In this process, the developers have a discussion with the end-users or the client to understand what they expect from the software.

- **Organizing Requirements –** Here, the developers arrange the requirements in the order of priority, importance and convenience.

- **Negotiation & Discussion –** At times**,** there may exist ambiguities or conflicts in requirements coming from different stakeholders. Such issues are discussed or negotiated with the stakeholders in this important process, for correctness and clarity. After a consensus has been received, realistic requirements are prioritised. At the same time, unrealistic requirements are reasonably compromised .

- **Documentation –** At this stage, all functional and non-functional , formal & informal, requirements are duly documented and are made accessible for the next phases of processing.

3.7.2 The Techniques of Requirement Elicitation

Different techniques are used in the process of requirement elicitation to discover the requirements. Some of these are mentioned below-

A . Interviews

An interview is a strong and widely used medium to collect user requirements. Different types of interviews employed by organizations are:

- **Structured or closed interviews**- Here, the interviewer decides about every single piece of information to gather well in advance. The matter of discussion and the pattern to follow is chalked out firmly.

- **Non-structured or open interviews**- Here, the interviewer does not decide on the information to gather in advance. Therefore, such interviews may be less biased and more flexible.

- **Written interviews**

- **Oral interviews**

- **One-to-one interviews**- Here, the interview is held between the interviewer and the interviewee, across the table.

- **Group interviews** - Here, the interview is held between categories of participants. As a number of people are involved in such interviews, group interviews have the capacity to unveil any missing requirement.

B. Surveys

The organization may collect required information by conducting specially designed surveys among the stakeholders. In the survey, queries are made about the expectation and requirements of the stakeholders from the proposed software system.

C. Questionnaires

In this method, a document is provided to all the stakeholders. It includes a pre-defined set of questions (in objective format, along with respective options). The response of each stakeholder is then collected and compiled.

However, this method has an important drawback. If a relevant option for a particular issue is not included in the questionnaire, there is every chance that the issue will be left unaddressed.

D. Task analysis

This technique may be used by the engineers and developers to analyze the functionality for which the proposed software system is needed. If the client has some similar software in place already to accomplish

certain operations, the same is studied and from this, the requirements of the proposed software system are collected.

E. Domain Analysis

Every software can be grouped into certain categories of domains. The experts in this particular domain can be consulted to analyze the general and specific requirements.

F. Brainstorming

In this method, an informal discussion cum debate is organized among the various stakeholders . Inputs coming from the stakeholders are recorded for further analysis by the organization.

G. Prototyping

Prototyping is all about building a user interface that is similar to the intended software. However, this interface does not include the detailed functionalities of the proposed software. But, from this, the user can interpret the proposed features of the software product. It helps the user to provide a better idea of his requirements. This method becomes very useful if no software is installed at the client's end for the reference of the developer and also when the client is not clear about his own requirements. In such cases, the developer creates the prototype depending on the raw and initial requirements coming from the user. The prototype is then provided to the client and his feedback is noted down. This feedback is considered as the input for requirement collection.

H. Observation

In this technique, an expert team visits the workplace or organization of the client. They observe and gather ideas about the actual working place where the existing systems are installed. They study the prevailing workflow at the end of the client end and the way the execution problems are managed. The team would draw important insights and conclusions which would form the functionalities expected from the proposed software.

3.8 REQUIREMENTS ANALYSIS

After the process of requirements gathering gets completed, the system analyst studies the gathered requirements in order to understand the exact requirement of the customer. This is done with a view to resolving any ambiguity in the customer's requirements.

STOP TO CONSIDER

The main aim of requirement analysis phase is to analyse the collected information in order to get a clear understanding of the system to be developed.

The following basic but important questions pertaining to the project should be clearly understood by the analyst, in advance:

• What is the problem?

• Why should we solve the problem?

• What are the possible solutions to the problem?

• What precisely are the data input to the system and what exactly are the data outputted by the system?

• What are the most likely complexities that might arise while trying to solving the problem?

• If the developed software needs to interface with any external software or hardware, then what exactly would be the format of data interchange with the external system?

STOP TO CONSIDER

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

Once the analyst gets a good understanding of the customers' requirements, he/she proceeds to resolve the following problems that he/she detects in the requirements.

a)**Ambiguity**

Since an unclear or ambiguous requirement can lead to incorrect software development, hence the analyst tries to resolve it.

b)**Inconsistency**

The analyst attempts to get the contradicting requirements given by say two customers resolved as this will hamper the development of precise software by the customer.

c)**Incompleteness**

The analyst gets the incompleteness in requirement solved by incorporating the requirements that have been overlooked by the customer. Several activities are involved in analyzing the requirements of the proposed software. Some of them are mentioned below :
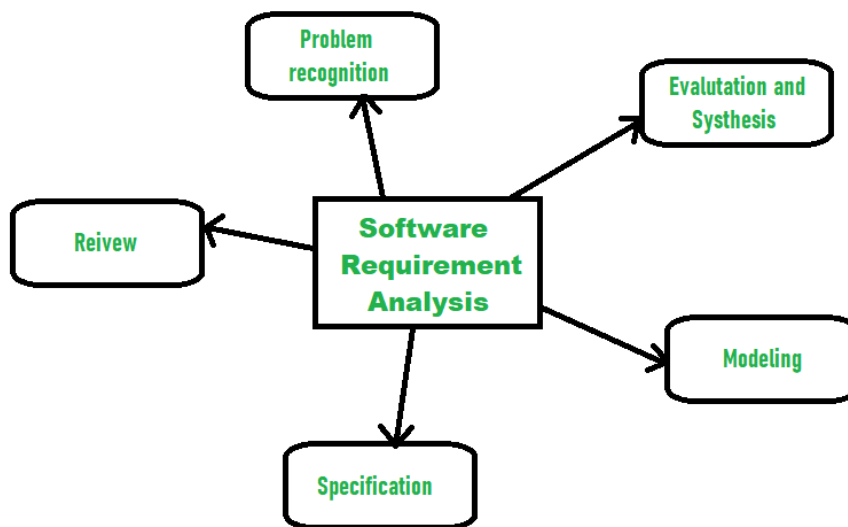
Fig:3.3 Activities involved in analyzing the requirements

3.8.1 Problem Recognition

The primary aim of the requirement analysis process is to completely understand the objectives of the requirement. It would also look into why the software is required, what value it will add to the product, what its benefits will be, if it increases the product quality and if it has any other effects. These points are studied thoroughly so that business problems can be fulfilled.

3.8.2 Evaluation and Synthesis

Evaluation means judging whether something is worth it or not. Whereas, synthesis refers to creating or forming something. The following tasks may constitute the process of evaluation and synthesis:

- To define all the necessary functions of the proposed software.

- To define the external data objects that are easily observable.

- To evaluate whether the data flow is worth it or not.

- To understand the overall working or behavior of the system

- To find out the constraints of the system.

- To understand the character of the system interface in order to thoroughly understand how a system interacts with other components or with other systems.

3.8.3 Modeling

After information gathering is complete with the help of the above tasks, the next step is to establish behavioral and functional models. Here, the behavior and function of the system are checked with the help of a domain model, also known as a conceptual model.

3.8.4 Specification

 Here, an SRS is developed to specify the requirements and to determine if these are functional or non-functional. The objective of software requirement is to specify the requirements of the software product in a consistent, unambiguous and concise manner- with the help of appropriate formal notations.

Specifying the software requirements is a critical step in software development. If not done in a proper manner, it may even lead to a crippled system that may be extremely difficult to rectify later on.

3.8.5 Review

After the SRS is developed, it should be reviewed. This is to check if the SRS can be improved or not. If there is scope for improvement, it must be refined to enhance the quality.

3.9 SRS OR SOFTWARE REQUIREMENT SPECIFICATION

After the analyst has collected all the required information regarding the software to be developed, and has removed all the anomalies, incompleteness and inconsistencies from the specification, he/she starts to organize the requirements, in systematic order, in the form of an SRS document. The SRS is the official document that contains what is required from the developers of the system. It comprises detailed user requirements and complete system specifications. According to Henninger, an SRS must satisfy the following six requirements:

➢ It should only specify the external system behavior

➢ It should specify the constraints related to the implementation.

➢ It should be adaptable to change.

➢ It should work as a reference for those who maintain the system.

➢ It should record the life cycle of the proposed software system.

➢ It should facilitate a standard and predictable response to undesired events.

SRS document is supposed to cater to the needs of a varied class of users some of which are mentioned below-

• Users, customers and marketing personnel

• Project Managers

• Software Developers

• Test Engineers

• User Documentation writers

• Maintenance Engineers

3.9.1 Need for SRS

The following points will help you understand the needs for SRS-

- SRS forms the base of the agreement between the supplier and the user. The user may not understand a single bit about software but, he needs to be satisfied. The developers will develop the software. But, he may not at all know about the problems the software must solve. SRS is the very medium that can bridge the communication gap.

- SRS can specify the needs of the user in a manner both the user and the developer can understand

- SRS can also help the user understand his needs better. Users may not always be clear about what to expect from the software. The SRS must analyze and apprehend the potential of the software to add value to the user and must enlighten the user about this.

- SRS provides a reference point for validating the final software product. It provides a clear understanding of what is expected from the software at the time of the validation.

- A high-quality SRS reduces the cost of software development

3.9.2 Characteristics of a Good SRS

Here are the desired characteristics of a good SRS:

1. **Concise:** An SRS should be concise. It should also be devoid of inconsistency and ambiguity. At the same time, it should also include each and every requirement specified by the user.

2. **Structured**: An SRS should be well structured so that its style and structure can be easily modified without disturbing the completeness and consistency.

3. **Black-box view:** SRS should only define what the system should do and abstain from stating how to do it. This means that the SRS should not discuss the internal implementation issues. The SRS report should view the proposed system as a black box.

It should only define the behavior of the system that is externally visible. This is the reason, why the SRS document is also termed as the black-box specification of a system.

4. **Verifiable**: An SRS is considered verifiable if every single requirement specified by it is verifiable. This means it follows a procedure to verify that the intended software meets the user requirement.

5. T**raceable**: An SRS is said to be traceable if each and every requirement specified by it can be uniquely attributed to a source.

6. **Response to undesired events:** It should include responses to exceptional cases**.**

3.9.2 Important Categories of Customer Requirement

While documenting users' requirements, utmost care should be taken to categorise and document the requirements in a different section of the SRS. The varied user requirements can be categorised in the following points

3.9.3. Functional Requirements

As discussed in 3.4.1 the functional requirements part discusses the functionalities required from the system. With a high-level function, the user should be able to accomplish some meaningful part of the work.

3.9.4 Non-Functional Requirements

As discussed in section 3.4.2 Non-functional requirements deal with the

characteristics of the system which cannot be expressed as functions.

3.9.5 Goals of Implementation

The goals of implementation part of the SRS documents some general suggestions regarding development. These suggestions guide trade-offs among design goals. This section can document issues with regards to revisions of the system functionalities that might be required in the near future, reusability issues, new devices to be supported in the future, etc. These are some of the items which the developers need to keep in their mind during development so that the developed system may meet these aspects that are not required at present.

3.9.6 Identify Functional Requirements

The functional requirements often need to be identified either from a conceptual understanding of the problem or an informal problem description document. Each high-level requirement is characterised by how a system is used by some users to perform certain meaningful jobs. A system can have different types of users and their requirements from the same system may also be completely different from one another. Hence, it is often necessary to identify the different levels of users who might use the system. Then it should make an attempt to identify each user's requirements from their perspective. Here we list all the functions {fi} performed by the system. Each function fi, as shown in the figure below (fig. 3.1), is considered as a conversion of a specified set of input data into the corresponding output data.

Example:

Consider the case of a library system, where –

 F1: Search Book details function (fig. 3.4)

Input: Author's name

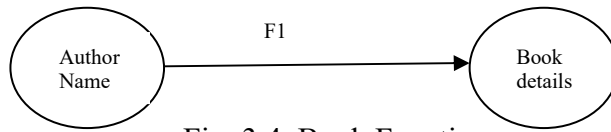Output: Details of the books of the author and their location inside the library

Fig. 3.4. Book Function

Therefore, the function Search Book (F1) accepts an author's name and converts it into book details.

Functional requirements literally describe a set of high-level requirements, with each high-level requirement taking some data from the user and thereby providing some data to the user as output. Also ,each high-level requirement itself might consist of many other functions.

3.9.7 Document Functional Requirements:

For the documentation of the functional requirements, we have to specify the set of functionalities that the system supports. A function is specified by identifying the state at which the data is inputted to the system, its input and output data domain, and the type of processing that is to be carried out on the input data to get the desired output data. Let us try to document the withdraw-cash function of an Automated Teller Machine(ATM) system. The withdrawal of cash **(**the withdraw-cash function**)** is a high-level requirement.   It has a number of sub-requirements corresponding to the different user interactions. These varied interaction sequences capture the different scenarios for an ATM .**An Example: Automated Teller Machine(ATM)**

Functional Requirements of an ATM

- Withdraw Cash

- Deposit Cash

- Balance Enquiry

- Passbook Update

- Transaction Details

- PIN Change

We will look at the functional requirements of Withdraw Cash from an ATM

Fig:3.5

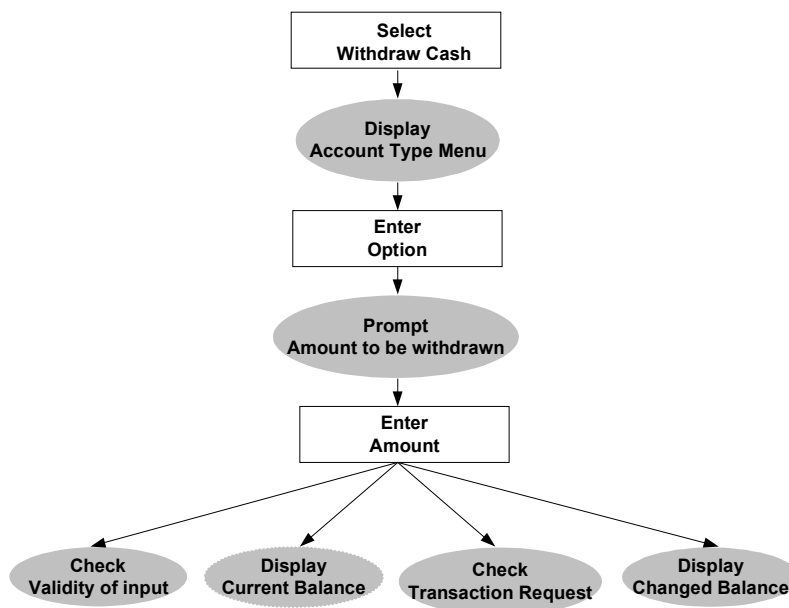**F1: Withdraw Cash**

Description: This function first ascertains the type of account the user has and the account number from which he attempts to withdraw cash . It verifies the balance to check if the amount requested is available in the account or not. If sufficient balance is available, it sends the required cash amount as output; else, it generates an error

F1.1: Select Withdraw Cash

Input: "Withdraw Cash" Option

Output: User is Prompted to fill in the Account Type

F1.2: Select Account Type

Input: User Option

Output: User is Prompted to enter the Amount

F1.3: Read required Amount

Input: Amount to be withdrawn in integer values in the range of more than 100 and below 10,000 and in multiples of 100

Output: Processing for "Valid Transaction" with requested cash and printed transaction OR "Failed Transaction" with a regret message **.**

3.9.8 Techniques for Representing Complex Logic

A good SRS document should be able to characterize the conditions under which different interaction scenarios take place. Sometimes these conditions may be complex and many alternative processing and interaction sequences may also exist. Two primary techniques to analyze and represent complex processing logic are available: decision trees and decision tables.

1. Decision Trees- It presents a graphic view of the processing logic associated with decision making and the corresponding actions undertaken. The edges of a decision tree denote conditions and the leaf nodes denote the actions to be taken on the basis of the outcome of testing the condition.

For Example,

Consider Library Membership Automation Software (LMS). Assume that it supports the following 3 options-

- New member

- Renewal, and

- Cancel membership

New member option Decision: When a user selects the 'new member' option, the software asks for member details like the member's name, phone number, address etc. **Action**: If the user enters proper information, the software would create a membership record for the particular member. Also, a bill is printed against the annual membership charge and the payable security deposit.

Renewal option Decision: If the user selects the 'renewal' option, the LMS asks for the membership number of the member and his name to verify if he is an authorised member. Action: If his membership is found to be valid, the membership expiry date gets updated and the annual membership bill for the user is printed. Else, an error message is shown on the screen**.**

Cancel membership option Decision: If the user selects the 'cancel membership' option, the LMS asks for the membership number of the member and his name to verify if he is an authorised member. Action:

The software cancels the membership. Also, a cheque bearing the balance amount due to the member gets printed. Finally, the membership record gets deleted from the database.

Decision tree representation of the above example The tree shown in the figure below, (fig. 34.3) is a graphical representation of the above-mentioned example. After the software gets the information from the user, it makes a decision and accordingly, performs the corresponding actions.
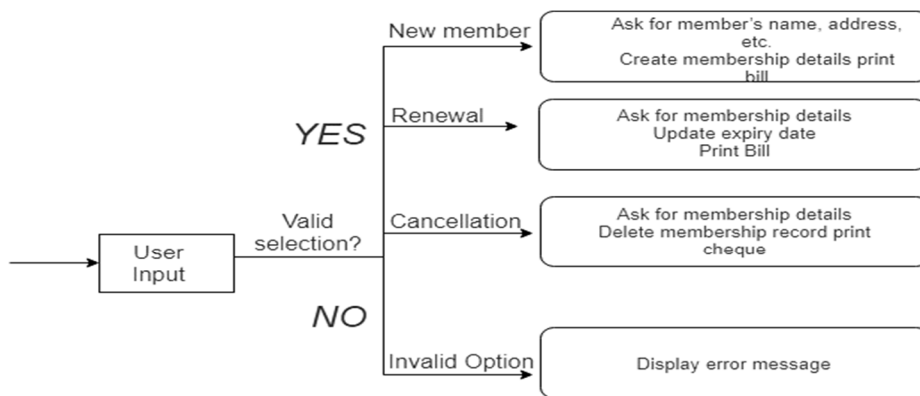
Decision tree for LMS

Fig 3.6

2. Decision Tables

Decision tables are used to represent the otherwise complex processing logic in a matrix or tabular form. The top rows of the table specify the conditions or variables to be evaluated. The rows at the bottom of the table specify the actions to be performed upon satisfaction of the corresponding conditions.

Example

Consider the LMS example discussed above. The decision table shown in the figure below (fig. 34.4) shows a way to represent the problem in a tabular form. The table here is divided into 2 parts. The part at the top shows the conditions and the part at the bottom shows the actions that have been taken. Each column of the table stands for a rule.

Conditions				
Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
Actions				
Display error message x	-	-	-	
Ask member's details	-	x	-	-
Build customer record	-	-	x	-
Generate bill	-	x	x	-
Ask member's details	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

Fig. 3.7 Decision table for LMS

From the table shown above, you can easily see that, if the valid selection condition becomes false, the action performed with reference to this condition is 'display error message' and so on.

3.9.9 Problems without an SRS Document

Here are some of the problems an organization will face without an SRS:

• If an SRS is not there, the software system implemented would not be able to address the needs of the customers.

• Software developers will not be sure about whether they have developed the software as per the customer requirements

• It will be extremely difficult for the maintenance engineers to understand the functionalities of the system.

• The document writers will find it very difficult to script the user manuals properly.

CHECK YOUR PROGRESS

1. Mark the following as True or False:

a. All software engineering principles are backed by either scientific basis or theoretical proof.

b. Functional requirements address maintainability, portability, and usability issues.

c. The edges of decision tree represent corresponding actions to be performed according to conditions.

d. The upper rows of the decision table specify the corresponding actions to be taken when an evaluation test is satisfied.

e. A column in a decision table is called an attribute.

2. i) How many feasibility studies is conducted in Requirement Analysis?
   a) Two
   b) Three
   c) Four
   d) None of the mentioned

ii). How many phases are there in Requirement Analysis ?
   a) Three
   b) Four
   c) Five
   d) Six

iii). Which one of the following is a functional requirement ?
   a) Maintainability
   b) Portability
   c) Robustness
   d) None of the mentioned

iv) . An SRS document normally contains

   a) Functional requirements of the system

   b) Module structure

   c) Non-functional requirements of the system

   d) Both a and b

v) Consider a system where a heat sensor detects an intrusion and alerts the security company. What kind of requirement the sysem provides?

   A). Functional
   B). Non-Functional

   C). Known

   D). None of the mentioned

3.10 SUMMING UP

This unit discusses various aspects of software requirements analysis, the significance of the use of engineering approach in software design, various tools for gathering the requirements and specifications of prototypes. Due to the complexity associated with software development, the engineering approach is being used in software design. The use of the engineering approach in the area of requirements analysis has evolved the field of Requirements Engineering. Before the actual system design commences, the system architecture is modelled.

3.11 ANSWERS TO CHECK YOUR PROGRESS

Answers for Q1:

  a)True

  b)False

  c)False

d)False

e)False

Answers for Q2:

i) b

ii) c

iii) d

iv)d

v) a

3.12 POSSIBLE QUESTIONS

1. Identify the important parts of an SRS document. Identify the problems an organization might face without developing an SRS document.
2. Identify the non-functional requirement-issues that are considered for a given problem description.
3. Discuss the problems that an unstructured specification would create during software development.
4. Why SRS document is often touted as a "Black Box" document?
5. "SRS document should be a flexible document" - Agree or disagree the comment
6. How Requirement Engineering is related to process development models?

3.13 REFERENCES AND SUGGESTED READINGS

- Software Engineering, Sixth Edition, 2001, Ian Sommerville; Pearson
- Education.
- Software Engineering – A Practitioner's Approach, Roger S. Pressman;
- McGraw-Hill International Edition.
- Fundamentals Of Software Engineering, 2014 4Th Edn. by Rajib Mall, PHI.

UNIT 4: SOFTWARE PROJECT PLANNING

Unit Structure:

4.1 INTRODUCTION

Software development is considered as a complex task involving processes, procedures and people. Therefore, for the successful development of a software an effective software management is required. Historically, software projects have the dubious distinction of

exceeding project schedule and cost. Estimating cost and duration is still a weak link in software project management. The aim of this unit is to give an overview of different project planning techniques and tools used by modern day software project managers. It is the responsibility of the project manager to make as far as possible accurate estimations of effort and cost. This is particularly what is desired by the management of an organization in a competitive world. This is especially true of projects subject to competition in the market where bidding too high compared with competitors would result in losing the business and a bidding too low could result in financial loss to the organization. This makes software project estimation crucial for project managers.

4.2 UNIT OBJECTIVES

After reading this unit, the reader will get a grasp of the following:

- Responsibilities of software project manager.

- The need and purpose of project planning.

- The project planning process.

- A project plan.

- Project scheduling.

- Techniques for project scheduling include Gantt and PERT chart.

- Project staffing.

4.3 RESPONSIBILITIES OF A SOFTWARE PROJECT MANAGER

Effective project planning is the key to successful completion of the software project and project manager is the person responsible for it. Software project managers is responsible for steering a project to success. It is difficult to describe the job responsibilities of a project manager. The responsibility of a project manager spans from invisible activities such as building up team morale to clearly visible activity like

customer presentations. Most managers take the responsibility for project proposal writing, scheduling, project staffing, project cost estimation, software process tailoring, software configuration management, project monitoring and control, risk management, managerial report writing and presentations, interfacing with clients, etc. These numerous activities are varied and difficult to enumerate, but it can be broadly classified into project planning, and project monitoring and control activities. The project planning activity is undertaken before development starts. While, the project monitoring and control activities are undertaken once the development activities start.

STOP TO CONSIDER

The different activities of a project manager can be classified into-
a) Project planning
b) Project monitoring and control activities

Before starting a software project, it is important to determine the various tasks to be performed. The tasks are then properly allocated among individuals involved in the process of software development. Thus, planning is essential as it results in effective software development.

Project planning is an well-structured and consolidated management process with consistent focus on activities that will lead to successful completion of the project. It helps in preventing obstacles that crop up in the project, as for example changes in organization's or project's objectives, non-availability of resources etc. Further, project planning also aids in optimal usage of the allotted time for a project and better utilization of resources. Some of the additional objectives of project planning are listed below.

- To define the roles and responsibilities of the members in a project management team.

- To determine project constraints.

- To check whether the user requirements along with schedule are feasible or not.

- To ensure that the works of project management team is aligned with the business objectives.

STOP TO CONSIDER

A project plan defines the project goals and objectives, defines tasks and the means to achieve the goals, identifies what resources will be needed and associated budgets and timelines for completion.

Several individuals work together for planning a project which include senior management and project management team. The senior management is responsible for employing the team members. The senior member team also provide resources that are required for the project. The project management team, which consists of project managers and developers, is generally in charge of planning, determining, and tracking the various activities of the project.

The following essential activities are carried out during project planning:

• Estimating the following attributes of the project:

Project size: What will be problem complexity, expressed in terms of the effort and time required to develop the product?

Cost: How much cost will be incurred to develop the project?

Duration: What is the duration for completing the software development?

Effort: How much effort would be required?

The success of the subsequent planning activities will depend on the accuracy of these estimations.

• Work break down structure

• Risk identification, analysis, and abatement planning

•Scheduling manpower and other resources

• Miscellaneous plans such as quality assurance plan, configuration management plan, etc.

STOP TO CONSIDER

Project planning is undertaken and completed even before any developmental activity starts.

The effectiveness of the following planning activities relies on the accuracy of those estimations.

- planning force and alternative resources
- workers organization and staffing plans
- Risk identification, analysis, and abatement designing

Miscellaneous arranges like quality assurance plan, configuration, management arrange, etc
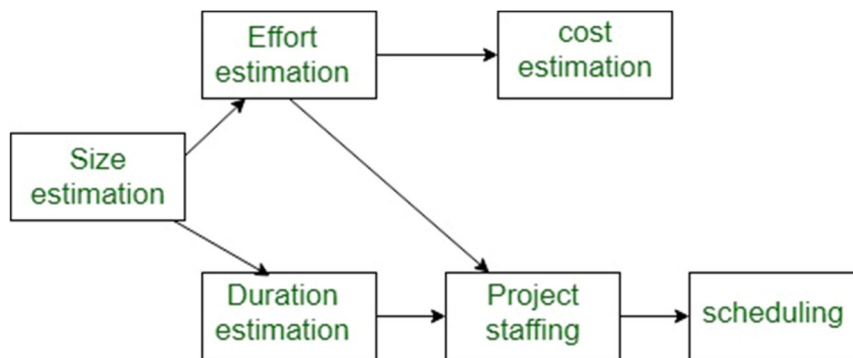
Precedence ordering among planning activities

Fig:4.1

Fig. 4.1 shows the order in which different project planning activities may be undertaken. It can be easily observed that size estimation is the first activity. It is also the most fundamental parameter based on which all other planning activities are carried out. Other estimations such as estimation of effort, cost, resource, and project duration are also very important components of project planning.

STOP TO CONSIDER
Size is the most fundamental parameter based on which all other parameters are estimated

4.5 SLIDING WINDOW PLANNING

Project planning needs due care and attention as adhering to unrealistic time and resource estimates leads to schedule slippage. Delays in

schedule can cause customer dissatisfaction and adversely affect the morale of the team. This may even lead to project failure. But project planning is a very challenging task. Especially it is very difficult to prepare accurate plan for very large projects. This is partly difficult because of the fact that the scope of the project, proper parameters, project staff, etc. may change during the span of the project. To overcome this problem, sometimes project managers undertake project planning in stages. Planning a project over a number of stages protects managers from making big commitments too early. This technique of staggered planning is known as Sliding Window Planning. In the sliding window technique, starting with an initial plan, the project is planned more accurately in successive development stages. At the start of a project, project managers have incomplete knowledge about the details of the project. Their information base gradually improves as the project progresses through different phases. After the completion of every phase, the project managers can plan each subsequent phase more accurately and with increasing levels of confidence.

4.6 SOFTWARE PROJECT MANAGEMENT PLAN (SPMP)

When project planning reaches its completion, project managers document their plans in the form of the Software Project Management Plan (SPMP) document. The list of various items that an SPMP document should discuss are mentioned below. This list can be referred to as a possible organization of the SPMP document.

Organization of the Software Project Management Plan (SPMP) Document

1. **Introduction**

    (a) Objectives

    (b) Major Functions

    (c) Performance Issues

    (d) Management and Technical Constraints

2. **Project Estimates**

(a) Historical Data Used

(b) Estimation Techniques Used

(c) Effort, Resource, Cost, and Project Duration Estimates

3. Schedule

(a) Work Breakdown Structure

(b) Task Network Representation

(c) Gantt Chart Representation

(d) PERT Chart Representation

4. Project Resources

(a) People

(b) Hardware and Software

(c) Special Resources

5. Staff Organization

(a) Team Structure

(b) Management Reporting

6. Risk Management Plan

(a) Risk Analysis

(b) Risk Identification

(c) Risk Estimation

(d) Risk Abatement Procedures

7. Project Tracking and Control Plan

8. Miscellaneous Plans

(a) Process Tailoring

(b) Quality Assurance Plan

(c) Configuration Management Plan

(d) Validation and Verification

(e) System Testing Plan

(f) Delivery, Installation, and Maintenance Plan

STOP TO CONSIDER
The SNMP is the end product of the planning process

4.7 PURPOSE OF PROJECT PLANNING

In order to accomplish a specific purpose, software project is carried out which can be categorized under two heads namely, project objectives and business objectives. Some of the most common **project objectives** are listed below.

- **Meet user requirements:**

  Understand the user requirements and develop the project accordingly.

- **Meet scheduled deadlines:**

  Complete the project milestones, as laid down in the project plan, on time so that the project gets completed according to the schedule.

- **Be within budget:**

  To manage the overall project cost in such a way that the project is completed within the allocated budget.

- **Produce quality deliverables:**

  Ensure that quality, accuracy and overall performance of the project is maintained.

Business Objectives

Business objectives plays the role for ensuring that the organizational objectives and requirements are accomplished in the project. In general, these objectives are related with business process improvements, quality improvements and customer satisfaction. Some of the most common project objectives are listed below.

- **Evaluate processes:**

  Evaluate the business processes and make changes as and when required during the progress of the project.

- **Renew policies and processes:**

In order to carry out the task effectively, flexibility to renew the policies and processes of the organization must be provided.

- **Keep the project on schedule:**

  Reduce the downtime (period of no work done) by effectively managing the factors, such as unavailability of resources during software development, that hampers the development.

- **Improve software:**

Use suitable processes so that the software developed meets the organizational requirements and the organization gains a competitive advantage.

STOP TO CONSIDER
Purpose of project planning is to accomplish project and business objectives

---

4.8 PROJECT SCOPE

Given the user requirements, the project management team decides the scope of the project before it begins. This scope furnishes a detailed description of features, functions, interfaces, and constraints of the software that needs to be considered. Functions gives a description of the tasks the software is expected to accomplish. Features describe the attributes the software should have as per the user requirements. Constraints express the limitations that are imposed on software by hardware, memory etc. Interfaces describe the interactions of modules and functions of software components with each other. Project scope also takes into account the software performance, which, again depends on its processing capability and response time required to produce the output.

Once the scope of the project is determined, it is crucial to understand it properly so as to develop the software aligned with the user requirements. After this, cost and duration of the project are estimated. In case, the project scope is not determined on time, the project may fail to complete within the specified schedule. Project scope gives details of the following information.

- The elements included in and excluded from the project

- The processes alone with the entities

- The functions and features need to be included in the software to meet the user requirements.

STOP TO CONSIDER

The project management and senior management team should communicate with the users to understand their requirements and develop software according to those requirements and expected functionalities.

4.9 PRINCIPLES OF PROJECT PLANNING

For the project to begin with well-defined tasks, project planning should be effective .An effective project plan helps to minimize any additional costs incurred on the project while it is in progress. For project planning to be effective, some principles are followed which are listed below.

- **Planning is necessary:**

   Planning should be carried out before a project begins. For it to be effective, objectives and schedules should be unambiguous and understandable.

- **Risk analysis:**

   Before starting a project, the senior management along with the project management team must consider the risks therein that may affect the project. As for instance, the user might want some changes in their requirements while the project is in progress. To tackle such a case, the time and cost estimation should be done accordingly (to meet the new requirements).

- **Tracking of project plan:**

   Once the project plan is ready, it should be tracked and modified accordingly.

- **Meet quality standards and produce quality deliverables:**

The project plan should be able to identify processes by means of which the project management team can ensure desired quality in software. Based on the selected process for ensuring quality, the time and cost for the project is estimated.

- **Description of flexibility to accommodate changes:**

    The final outcome of project planning is in the form of a project plan, which should be flexible enough to allow changes to be incorporated when the project is in progress

4.10 PROCESS OF PROJECT PLANNING

The project planning process comprises of a series of interlinked activities followed in an ordered sequence in order to implement user requirements. It includes the elucidation of a series of project planning activities along with individual(s) responsible for performing these activities. Furthermore, the project planning process consist of the following.

1. The objectives and scope of the project.
2. Name of techniques used to perform project planning
3. Effort of individuals (expressed in time) involved in the project.
4. Resources required for the project
5. Project schedule and milestones
6. Risks associated with the project.

The process of project planning comprises of several activities which are crucial for carrying out a project in a systematic manner. These activities consist of series of tasks undertaken over a period of time in the process of developing the software. These activities comprise of estimation of effort, time, and resources required and risks associated with the project.
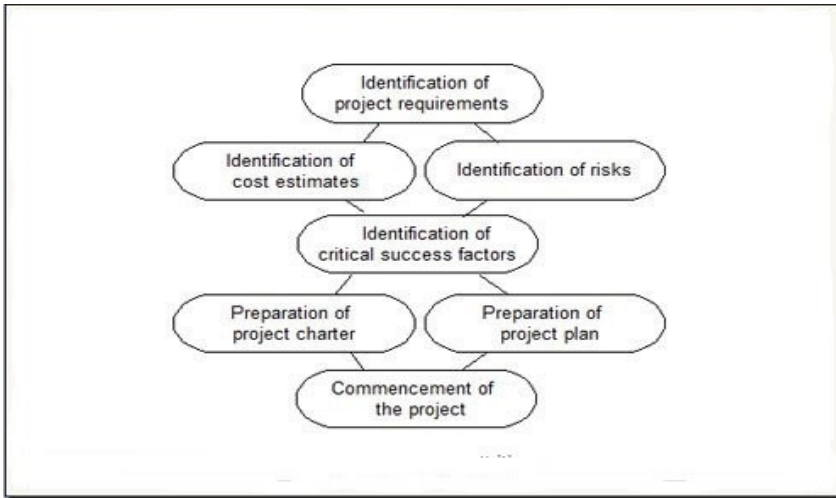
Fig 4.2

Following activities are performed during the project planning process

- **Identifying the project requirements:**

    Before starting a project, it is important to identify the requirements of the project because it is the identification of project requirements that will help in performing the project related activities in a systematic manner. These requirements consist of information such as project scope, data and functionality required in the software, and roles determined for the project management team members.

- **Identifying cost estimates:**

    Along with the effort and time estimation, it is essential to estimate the cost that the project will incur. The cost of hardware, the cost required for the maintenance of hardware components and the cost of network connections are included in the cost estimation. In addition to this, cost for the individuals involved in the project is also estimated.

- **Identifying risks:**

    Risks can be defined as unexpected or undesired events that have an adverse effect on the project. A software project may encounter several risks (such as technical risks and business risks) that hamper the project schedule and multiply the cost of the project.

Hence identifying risks before starting a project, helps in understanding their probable magnitude of impact on the project.

- **Identification of critical success factors:**

  For the success of a project, critical success factors need to be recognised. These factors include the conditions that ensure greater chances of success for a project. Commonly, these factors include appropriate budget, support from management, proper schedule, and skilled software engineers.

- **Preparation of project charter:**

  A project charter gives a brief description of the project scope, time, quality, cost, and resource constraints as described during project planning. The management prepares it for approval from the sponsor of the project.

- **Preparation of project plan:**

  A project plan gives a description regarding the resources that are available for the project, individuals involved in the project, and the schedule according to which the project is to be carried out.

- **Commencement of the project:**

  After the project planning is complete and resources are allocated to team members, the software project commences.

After the determination of project objectives and business objectives, end date for the project is fixed. The project management team is entrusted with the responsibility of preparing the project plan and schedule aligned with the end date of the project. After the project plan is analyzed, the project manager conveys the project plan and its end date to the senior management. From time to time, the progress of the project is reported to the management. In the same way, the senior management is informed when the project is complete. In case there is any delay in completing the project, the project plan is re-analyzed and corrective actions are taken to complete the project. The project is tracked on a regular basis and in case of any modification in the project plan, the senior management is informed.

STOP TO CONSIDER

Once the project objectives and business objectives are determined for the project, its end date is fixed.

4.11 PROJECT PLAN

A project plan is the outcome of project planning. It provides information regarding the end date, milestones set, activities and deliverables of the project. Moreover, it also describes the duties and responsibilities of the project management team. The resources required, including the description of hardware and software (such as compilers and interfaces) , for the project are also finds place in the project plan. The description and lists of the methods and standards to be used are also mentioned in the project plan. These methods and standards include algorithms, tools, review techniques, design language, programming language, and testing techniques.

It helps a project manager to understand, monitor, and control the development of software project. This plan is acts as a means of communication between the users and project management team. There are a number of advantages associated with a project plan, some of which are listed below.

- It ensures that the software is developed as desired by the user (user requirements), objectives set, and scope of the project.

- It helps in identifying the role of each member of the project management team who are involved in the project.

- It helps to monitor the progress of the project as laid in the project plan.

- It enables to determine the available resources and the activities to be undertaken during software development.

- It gives an overview of the costs of the software project, which was estimated during project planning, to the management,

STOP TO CONSIDER

It should be noted that depending on the kind of project and user requirements there are differences in the contents of two project plans.

Generally, project plan is divided into the following sections.

- **Introduction:**

  It describes the objectives of the project and gives detail about the constraints that might affect the software project.

- **Project organization:**

  Illustrates about the responsibilities assigned to the team members of project management in order to complete the project.

- **Risk analysis:**

  Describes the probable risks involved during software development It also explains means to assess and reduce the effect of risks.

- **Resource requirements:**

  Specifies the software and hardware required to build the software project. Cost estimation is thus done according to these resource requirements.

- **Work break down:**

  Specifies the activities the project is composed of. It also describes the milestones and deliverables of the different project activities.

- **Project schedule:**

- Specifies how the activities are dependent on each other. Based on this, the time required by the project management team members to complete the project activities is estimated.

In addition to the above sections, there are a number of plans that may be a part of or related to a project plan. Some of these plans include quality assurance plan, verification and validation plan, configuration management plan, staffing plan, and maintenance plan.

STOP TO CONSIDER

Project Plan is the document containing information regarding the end date, milestones set, activities and deliverables of the project.

4.11.1QUALITY ASSURANCE PLAN

The quality assurance plan specifies the strategies and methods that needs to be followed in order to accomplish the following objectives.

- Ensuring that the project is developed, managed, and implemented in an organized way.

- Ensuring that the deliverables of the project is of acceptable quality before they can be delivered to the user.

4.11.2 Verification and Validation Plan

The verification and validation plan gives detail about the approach, resources and schedule to be used for system validation. It comprises of the following sections.

4.11.2.1 General information

It provides description of the aim or purpose of the system, scope, system overview, project references, abbreviations and acronyms, and points of contact. **Purpose** gives a description of the procedures to verify and validate the various components of the system. **Scope** provides information regarding the procedures to verify and validate as they relate to the project. **System overview** specifies information about the organization responsible for the project. Other information such as system name, system category, operational status of the system, and system environment are also specified by system overview. **Project references** gives the list of references used while preparing the verification and validation plan. **Acronyms and abbreviations** specifies the list of terms used in the project plan. **Points of contact** is meant to provide assistance to users, by the organization, as and when they encounter problems such as troubleshooting and so on.

4.11.2.2 Reviews and Walkthroughs

It is meant to provide information about the schedule and procedures. **Schedule** describes the end date to achieve the milestones set for the project. **Procedures** are the tasks that are associated with reviews and

walkthroughs. Each team member is responsible for reviewing the documents for errors and also ensure it is consistent with the project requirements. During walkthroughs, the project management team checks to see the correctness of the project as laid down in the software requirements specification (SRS).

4.11.2.3 System Test Plan and Procedures

Provides information related to the strategy used for testing the system, database integration, and platform system integration. System test strategy gives an overview of the various components required in order to integrate of the database and ensure that the application executes on at least two specific platforms. Database integration describes about how the database is connected to the GUI (Graphical User Interface). Platform system integration is a procedure which is performed on different operating systems with a view to test the platform.

4.11.2.4 Acceptance Test and Preparation for Delivery

It describes the procedure, criteria for acceptance, and installation procedure. Procedure illustrates on how acceptance testing needs to be performed on the software in order to verify its usability as it was expected to be. Acceptance criteria specifies the conditions based on which the software will get accepted. These includes those tests must be on all the components, features and functions and the system integration test. In addition to this, acceptance criteria also checks whether the software meets user expectations such as its ability to operate on several platforms. Installation procedure describes the steps on how to install the software in the specific operating system being used.

4.11.3 Configuration Management Plan

The configuration management plan defines the process, which is to be used while bringing about some changes in the project scope. Generally, it is concerned with redefining the objectives of the project

and its deliverables (software products that are delivered to the user after completion of software development).

4.11.4 Maintenance Plan

The maintenance plan lists the resources and processes that is required for making the software operational after its installation. Sometimes, software development team (or the project management team) is not entrusted with the responsibility of maintenance of the software. In such cases, a separate team known as software maintenance team is assigned the task of software maintenance.

The maintenance plan, comprises of the following sections.

a) Introduction and background:

It describes the services required by the software and the manner in which the software is to be maintained. It also defines the scope of al the maintenance activities that needs to be performed.

b) Budget:

Specifies the budget that will be required for carrying out operational activities and software maintenance.

c) Roles and responsibilities:

Defines the roles and responsibilities of the each of the team members which are associated with software maintenance and operational activities. It also mentions the skills that is required to perform maintenance and operational activities. Apart from the software maintenance team, software maintenance also comprises of the user support staff, user training staff, and support staff.

d) Performance measures and reporting:

It recognises the performance measures required in order to carrying out software maintenance. Moreover, it also describes how the different measures, essential for enhancing the performance of services (for the software), are recorded and reported.

e) Management approach:

It identifies the methodologies required for demonstrating maintenance priorities of the project. Because of this purpose, the management either

identifies new methodologies or refers to the existing methodologies. Apart from this management approach also describes the various levels at which the users are involved in software maintenance and operational activities. It also specifies the manner in which the users and the project management team will communicate with each other.

f) Documentation strategies:

Describes in detail the documentation prepared for user reference. Generally, reports, error messages, information about problems occurring in software, and the system documentation are included in the documentation.

g) Training:

It provides information regarding the training activities.

h) Acceptance:

It defines a point of agreement between the project management team and software maintenance team after the completion of implementation and transition activities. Once the agreement has been made, the software maintenance begins.

4.11.5 Staffing Plan

The staffing plan specifies the number of individuals required for a project. It includes selection and assignment of tasks to the members of the project management team. It also specifies the appropriate skills required to manage the project and to perform tasks so as to produce the project deliverables. In addition to this, Staffing Plan also provides information about the resources such as tools, equipment, and processes used by the project management team.

A staff planner undertakes the task of staff planning. He is the one who is responsible for determining the individuals that are available for the project. Moreover, a staff planner has a number of responsibilities which are listed below.

1) The staff planner identifies the individuals, who can be either from existing staff, staff on contract, or newly employed staff, that can be employed in the project. It is essential for the staff planner to know the

structure of the organization in order to determine the availability of staff.

2) The staff planner is responsible for determining the skills required to execute the various levels of tasks as mentioned in the project schedule and task plan. In case where staff with required skills is not available, the staff planner has to inform the project manager about the requirements.

3) The staff planner must ensure that the required staff with desired skills is available at the right time. In order to fulfill this requirement, the staff planner plans the availability of staff once the project schedule is fixed. For instance, the initial stage of a project require only the project manager along with a few software engineers as the staff whereas during software development phase, the staff will consist of software designers as well as the software developers.

4) Roles and responsibilities of the project management team members are also defined by the staff planner, so that, according to the tasks assigned to them, they can communicate and coordinate with each other. Depending on the size and complexity of the project the project management team can be further broken down into sub-teams

The staffing plan comprises of the following sections.

1) General information:

Specifies name of the project and the project manager, who is responsible for the project. It also specifies the start and end dates of the project.

2) Skills assessment:

Provides information, which is essential for assessment of skills. These information include the knowledge, skill, and ability of team members who are responsible for achieving the objectives of the project. Moreover, it specifies the number of members that will comprise a team, for the project.

3) Staffing profile:

It describes the staff profile required for the project. The profile defines the calendar time, the individuals involved, and the level of commitment. Calendar time specifies the period of time, expressed in

terms as month or quarter, that the individuals are required to complete the project. All the individuals involved in the project have specific designations such as project manager and developer. Level of commitment is the utilization rate of individuals such as work performed on full-time and part-time basis.

4) Organization chart:

It illustrates the organization of team members in the project management team. It also includes information such as name, designation, and role of each team member.

STOP TO CONSIDER
The staffing plan specifies the selection and assignment of tasks to the members of the project management team

---

4.12 PROJECT SCHEDULING

Project scheduling comprise one of the most important part in project planning activity. It gives details regarding the project start date and termination date, milestones set, and the tasks for a particular project. Moreover, it specifies resources in terms of person to be engaged, equipments and facilities to be used. A proper project schedule prepared aligned to the project plan not only aims to complete the project on the scheduled time but also ensures that no additional cost is incurred in the event of any delay in the project. a software project manager follows the following principles while preparing the project schedule:

1) Compartmentalization:

Modularize the project to form subtasks. The main purpose of compartmentalization is to make the project manageable.

2) Interdependency:

Identify the interdependency among the various tasks. All the activities may not be interdependent. But there are some activities which cannot be started until the activity on which it is dependent completes. on interdependent activities can be started simultaneously.

3) Time Allocation:

Determine the time required to complete each activity so that it can be allocated to each of the team members responsible for carrying out that particular job. Moreover, the members of the project management team should be assigned a start and end date on the based on the manner on which the work will be conducted (i.e., full-time or part time basis)

4) Effort Validation:

Ensure that the number of project team members allocated to a specific task conforms to the effort required for that particular task. This is specifically because every project management team has a specified number of members in it and if more or less members are allocated to a particular activity than required then the project may not complete on time.

5) Defined responsibilities:

Demarcate the roles and responsibilities of each of the members in the project management team. Thus, tasks should be allocated suited to their skills and abilities.

6) Defined outcomes:

The outcome of each and every task should be well defined. Generally, outcomes are defined in terms of product which are combined together to form the deliverables.

7) Defined milestones:

Set milestones for the completion of the product and its review for quality check.

STOP TO CONSIDER

Several aspects that are important in project scheduling are: a) Techniques of project scheduling b)Task Network c)Tracking the schedule

---

The first and foremost step in project scheduling is identification of the various tasks essential for completion of the project. The next step in scheduling is to breakdown a large task into a logical set of small

activities which can be assigned to different developers. After the the tasks has been broken down to smaller tasks and the work breakdown structure has been created, the dependency among the activities needs to be identified. Determining the dependency among the different activities determines the order in which the different activities would be carried out. For instance if an activity A needs the results of another activity B, then activity A must be scheduled after activity B. Generally, task dependencies define a partial ordering among the different tasks, i.e. one task may precede a subset of other tasks, but for some tasks there may not exist any precedence ordering defined between them (called concurrent task). Activity Network is used to represent the dependency among the activities

Once the activity network representation has been worked out, resources are allocated to each activity. Allocation of resource is generally done using a Gantt chart. After resource allocation is done, a PERT chart representation is developed to monitor and control the project. For task scheduling, the project tasks is decomposed into a set of activities. The time frame for each activity to be performed is determined. The end of each activity is called milestone. The progress of a project is tracks by the project manager by monitoring the timely completion of these milestones. In case he observes that the milestones started getting delayed, then he has to carefully control the activities, so that the overall deadline can still be met.

4.13 TECHNIQUES FOR PROJECT SCHEDULING

We need to use scheduling techniques in a project to align all its aspects so as to work corresponding to each other. A schedule should be proportionate with the time set for the project and all its resources should be used in an optimum manner. Given the variable nature of the project and its scope, it is hard to plan it, but the project management team is expected to do it otherwise they will be held responsible for it.

A schedule consists of all the activities included in the implementation and execution of a project within the pre-determined time frame of the project. A project schedule helps in prioritizing work involved in a project and finish it off in an orderly manner. It also helps in appointing the right person for the job and in the proper allocation of

the available resources. Time management and adjustments with the scope of a project is only possible if there is a proper schedule prepared for the project being worked upon

Numerous techniques have been in use for the purpose of project scheduling. These techniques are applied after every information is gathered from the project planning activities.

The most common techniques used for project scheduling are Activity network and critical path, Gantt Chart and PERT chart.

STOP TO CONSIDER

Work Breakdown Structure (WBS) is the procedure for decomposing a given task set recursively into small activities.

4.13.1 Activity Networks

Work Breakdown Structure (WBS) representation of a project is transformed into an activity network by representing the activities identified in WBS along with their interdependencies. An activity network displays the different activities making up a project, their estimated durations, and interdependencies (as shown in fig. 4.4). Each activity is represented by a rectangular node and the duration of the activity is shown alongside each task.

The Work Breakdown Structure of an MIS (Management Information System) problem is shown in the fig 4.3

STOP TO CONSIDER

Activity Network is used to represent the dependency among the activities

---
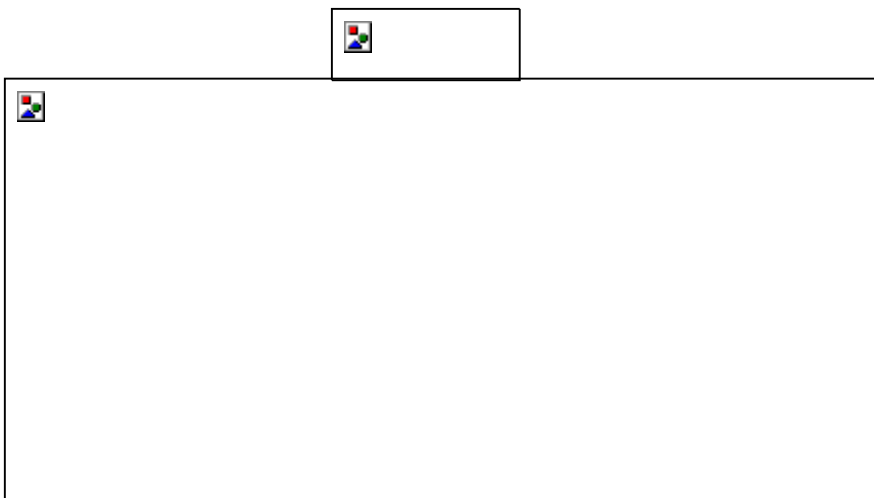
Fig 4.3 showing the WBS of an MIS problem.

The activity network representation of the same problem as shown in fig 4.3 is shown in fig 4.4
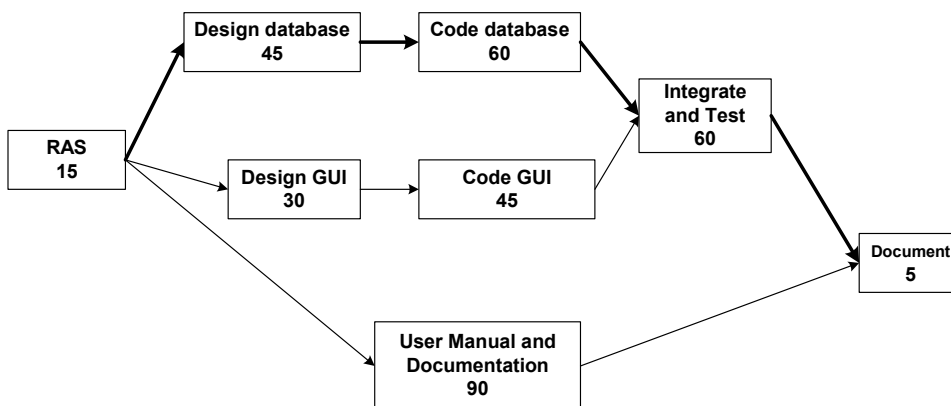
Fig 4.4 showing the Activity Network of the MIS problem.

Time durations for the different tasks can be estimated in several ways. One possible way that project managers use is that they empirically assign durations to different tasks. However, this is not a good technique for time estimation, as software engineers often do not agree to this unilateral decision. An alternative to this is to let engineer himself assess the time required for an activity he could be assigned to. Nevertheless, some managers prefer to estimate the time for various activities themselves. Many believe that an aggressive schedule could motivate the engineers to carry out his job in a better and faster way.

However experiments reveal that unrealistically aggressive schedules not only cause engineers to compromise on quality aspects, but also can lead to schedule delays. A better way to estimate the durations of task accurately is to let people involved in the project set their own schedules.

4.13.2 Critical Path Method (CPM)

The Critical Path Method is a technique for determining the activities with least scheduling flexibility known as critical activity. It should be noted that a delay in these activities will eventually delay the entire project. After the determination of these activities CPM defines the project schedule aligned to the activities that lie on the critical path method

From the activity network representation following analysis can be made. The minimum time (MT) to complete the project is the maximum of all paths from start to finish. The earliest start (ES) time of a task is the maximum of all paths from the start to the task. The latest start time is the difference between MT and the maximum of all paths from this task to the finish. The earliest finish time (EF) of a task is the sum of the earliest start time of the task and the duration of the task. The latest finish (LF) time of a task can be obtained by subtracting maximum of all paths from this task to finish from MT. The slack time (ST) is LS – EF and equivalently can be written as LF – EF. The slack time (or float time) is the total time that a task may be delayed before it will affect the end time of the project. The slack time indicates the "flexibility" in starting and completion of tasks. A critical task is one with a zero slack time. A path from the start node to the finish node containing only critical tasks is called a critical path. These parameters for different tasks for the MIS problem are shown in the following table.

Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code database	60	165	60	165	0
Code GUI part	45	90	120	165	75
Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code database	60	165	60	165	0
Code GUI part	45	90	120	165	75

The critical paths are all the paths whose duration equals MT. The critical path in fig. 4.4 is shown with a thicker arrow.

STOP TO CONSIDER

A path from the start node to the finish node containing only critical tasks is called a critical path.

4.13.3 Gantt Chart

Gantt charts are mainly used to allocate resources to activities. The resources allocated to activities include staff, hardware, and software. Gantt charts (named after its developer Henry Gantt) are useful for resource planning. A Gantt chart is a special type of bar chart where each bar represents an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity.

Gantt charts are used in software project management are actually an enhanced version of the standard Gantt charts. In the Gantt charts used for software project management, each bar consists of a white part and a shaded part. The shaded part of the bar shows the length of time each

task is estimated to take. The white part shows the slack time, that is, the latest time by which a task must be finished. A Gantt chart representation for the MIS problem of fig. 4.4 is shown in the fig. 4.5.
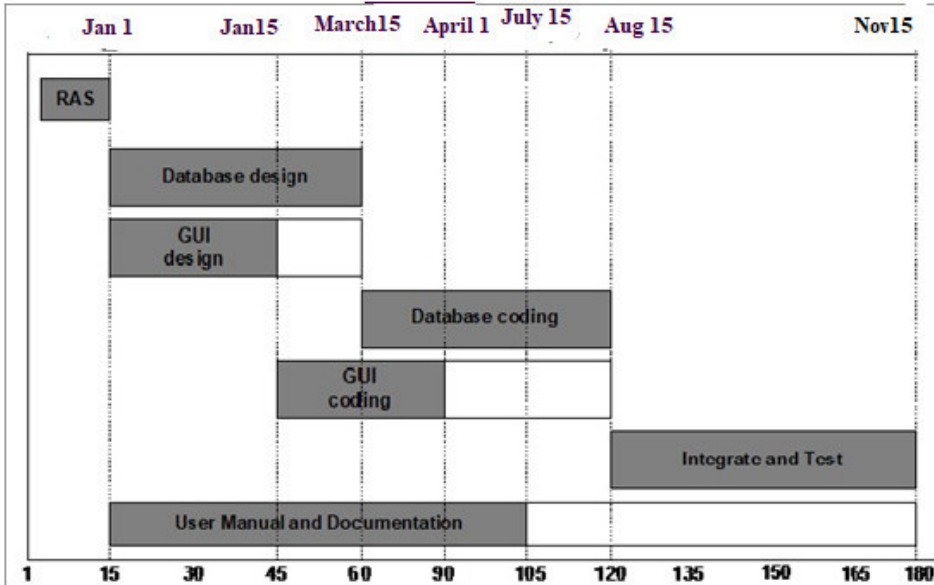
Fig. 4.5: Gantt chart representation of the MIS problem

STOP TO CONSIDER

Gantt charts are used to allocate resources to activities.

4.13.4 PERT Chart

PERT (Project Evaluation and Review Technique) charts consist of a network of boxes and arrows. The boxes represent activities and the arrows represent task dependencies. PERT chart represents the statistical variations in the project estimates assuming a normal distribution. Thus, in a PERT chart instead of making a single estimate for each task, pessimistic, likely, and optimistic estimates are made. The boxes of PERT charts are usually annotated with the pessimistic, likely, and optimistic estimates for every task. Since all possible completion times between the minimum and maximum duration for every task has to be considered, there are not one but many critical paths, depending on the permutations of the estimates for each task. This makes critical path analysis in PERT charts very complex. A

critical path in a PERT chart is shown by using thicker arrows. The PERT chart representation of the MIS problem of fig. 4.4 is shown in fig. 4.6. PERT charts are a more sophisticated form of activity chart. In activity diagrams only the estimated task durations are represented. Since, the actual durations might vary from the estimated durations, the utility of the activity diagrams are limited.
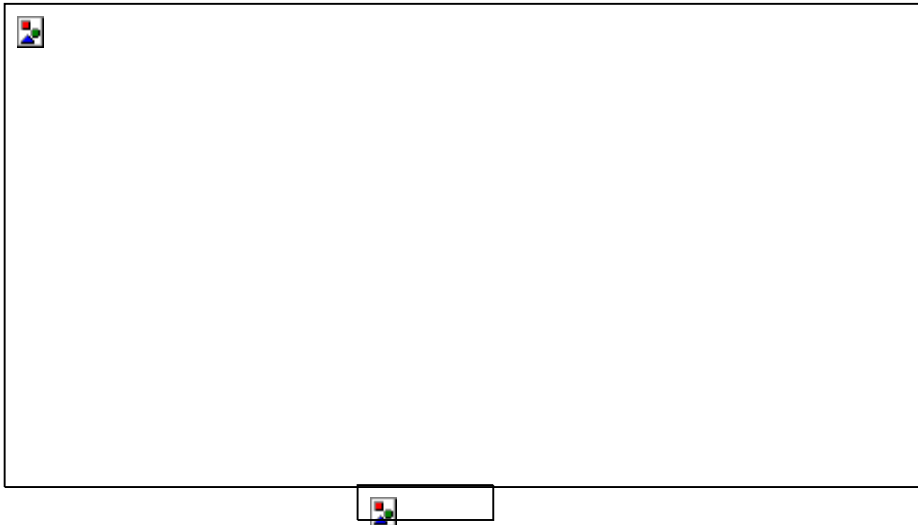
Fig. 4.6: PERT chart representation of the MIS problem

Gantt chart representation of a project schedule is helpful in planning the utilization of resources, while PERT chart is useful for monitoring the timely progress of activities. Also, it is easier to identify parallel activities in a project using a PERT chart. Project managers need to identify the parallel activities in a project for assignment to different engineers.

| **STOP TO CONSIDER** |
| PERT chart is used to monitor and control the projects. |

CHECK YOUR PROGRESS

A. Choose the correct option from the following:

1. Which of the following activity is undertaken immediately after feasibility study and before the requirement analysis and specification phase?

     a) Project Planning

     b) Project Monitoring

     c) Project Control

     d) Project Scheduling

2. This activity is undertaken once the development activities start?

     a) Project Planning

     b) Project Monitoring and Control

     c) Project size estimation

     d) Project cost estimation

3. In the project planning, which of the following is considered as the most basic parameter based on which all other estimates are made?

     a) Project size

     b) Project effort

     c) Project duration

     d) Project schedule

4. Once project planning is complete, project managers document their plan in

     a) SPMP document

     b) SRS document

     c) Detailed Design document

     d) Excel Sheet

5. Which of the Following method is not used as project scheduling technique

       a) Activity Diagram

       b) CPM

       c)Timesheet

       d) Gantt chart

b) Fill in the blanks:

1) The estimation that is carried out first by a project manager during project planning is _____.

2) Sliding Window Planning involves _____

3) Normally software project planning activity is undertaken

_____

4) _____ is the process of dividing the project into tasks and logically ordering them into a sequence.

5)Techniques used for project scheduling include _____, _____ and CPM diagram.

4.14 SUMMING UP

This unit discussed the about Project planning which is an well organized process which aims at successful completion of the software project. The main purpose of project planning is to accomplish business and project objectives. Fulfillment of user requirement, completion of the project within the scheduled timeframe and with the allocated budget and incorporation of quality in the software constitute the project objectives. Business objectives include evaluation of processes and also renewal and evaluation of policies and processes. Project scope highlights on the limitations of the project. The process of project planning consists of a set of related activities carried out in an orderly manner with a view to implement user requirements in the software. It also mentions the individuals responsible for each activity. The end product of project planning is the project plan which contains the end date, activities, milestones and deliverables. Project scheduling determines the time limit required for completing the project. Different

techniques such as CPM, Gantt Chart and PERT chart are used at different levels for the purpose of project scheduling.

4.15 ANSWERS TO CHECK YOUR PROGRESS

A) Multiple choice questions

1.(a)

2.(b)

3.(a)

4.(a)

5.(c)

B) Fill in the blanks

1. Size estimation

2**.** Planning progressively as development proceeds.

3. Before the development starts to plan the activities to be undertaken during development

4.Work Breakdown Structure

5.Gantt chart, PERT chart.

4.16 POSSIBLE QUESTIONS

1) List the important items that a Software Project Management Plan (SPMP) document should discuss

2) What do you understand by Sliding Window Planning? Explain using a few examples the types of projects for which this form of planning is especially suitable. What are its advantages over conventionalplanning?

3) Planning and Scheduling consume a lot of time. What will happen if software project commences without planning and scheduling.

4) List the important items that a Software Project Management Plan (SPMP) document should discuss.

5) When does the software planning activity start and end in software life cycle? List some important activities that a software project manager performs during software project planning.

4.17 REFERENCES AND SUGGESTED READINGS

- Software Engineering Principles and Practices, Second Edition, 2011, Rohit Khurana; Vikash Publication.
- Software Engineering – A Practitioner's Approach, Roger S. Pressman; McGraw-Hill International Edition.
- Fundamentals Of Software Engineering, 2014 4th Edition by Rajib Mall, PHI

UNIT 5: SOFTWARE DECOMPOSITION AND COST ESTIMATION TECHNIQUES

Unit Structure:

5.1 INTRODUCTION

In this unit we will learn about the software decomposition and its importance in project size estimation. The project resources plays an important role for successful development and completion of project.

We will also learn that the effective software project management is crucial to the success of any software project. In this unit we will learn that for the accurate estimation of the problem size is fundamental to satisfactory estimation of other project parameters such as effort, time duration for completing the project and the total cost for developing the software. The unit covers the empirical, Heuristics and Analytical size estimation techniques. The COCOMO model is discussed in detail.

5.2 UNIT OBJECTIVES

After completing the unit, you will be able to:

- The software resources, its type and Importance.

- Understand the importance of decomposition in software development.

- The concept of project planning and software project management techniques.

- The stages in the project planning and the different cost estimation techniques.

- The different size estimation techniques like Expert Judgement and Delphi Cost estimation Technique.

- Understand the COCOMO Model postulated by Boehm.

5.3 SOFTWARE RESOURCES

The software resources or the project resources are required for successful development and completion of project. These resources can be capital, people, material, tool or supplies that are helpful to carry out certain task in project. Without these resources it is impossible to complete project. In project planning phase identification of resources that are required for completion of project and they will be allocated is key element and very important task to do. There are mainly three types of resources that are considered and are very essential for execution of project and completion for project on time and budget. These resources can be represented by a pyramid called Resource Pyramid.

When software planner wants to specify resources, they specify resources, they specify it using four characteristics

➢ Description of resource

➢ Resource Availability

➢ Time of Resource when it will be available

➢ Duration of resource availability

People

_____

Reusable software

Components

_____

Hardware and Software

Tools

RESOURCE PYRAMID
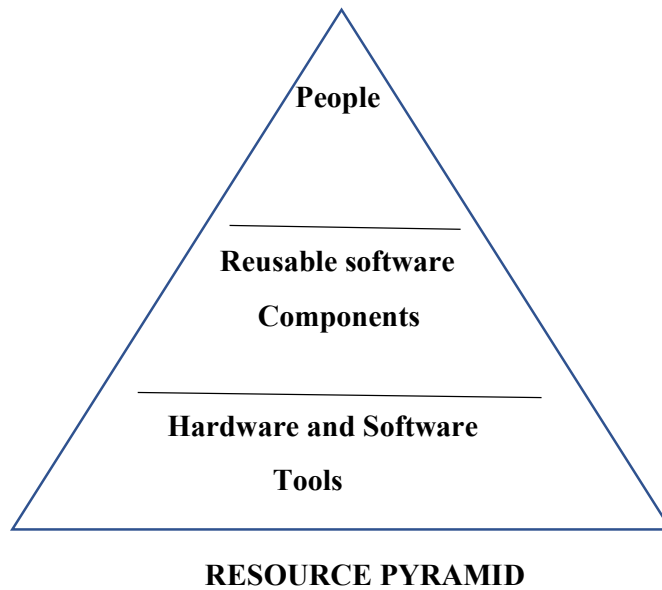
5.3.1 Types of Software Resources

There are mainly three types of resources:

a)   **Human Resource:** People or Human resource playa an important role in software development process. No matter what size is and how much complexity is there in the project, if you want to perform project task in an effective an effective manner, then human resources are very essential. In software industry the people are performed some organizational positions such as manager, software developer, software developer, and test engineers and so on that depends on their skills and specialty. For small project a single individual can perform all these activities, but for large project a big team of people are needed.

b)   **Reusable Components:** In order to accelerate and bring ease to the software development process the industry prefers to use some

Space for learners:

ready components of software. It can be defined as the software building blocks that can be created and reused in software development process. Managing budget is one of the most important task that all project managers have to do. The reusable resources helps in reducing the overall cost of software development. The use of component emphasizes reusability and is termed as Component based Software Engineering.

c)   **Hardware and Software Tools:** These are the actual material resources that are part of project. This type of resources should be planned before starting development of project otherwise it may cause problems for the project.

5.4 SOFTWARE DECOMPOSITION

Decomposition Technique uses the concept of "Divide and conquer" method for the cost estimation of a project. By decomposing the projects into major functions and related engineering activities the cost and size estimation can be performed in stepwise fashion. In the complete software development, the system is divided into major modules or the functional requirement of the system and then is further decomposed into simpler forms until the problem is solved using an algorithm. **Decomposition** in computer science, also known as **factoring**, is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain. The Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience (Historical Data) and takes the form :

d = $f(V_i)$,where d is one of the estimated values ie effort , cost and project duration.

V_i are selected as independent Line of code (LOC) or Function Point (FP).

Software project estimation is a form of problem solving and in most cases the   problem to be solved ie.(developing an cost and effort estimate for software project) is too complex to be considered in one piece. For this reason we decompose the problem, re characterize it as a set smaller and hopefully more manageable problems.

Line of code (LOC) and Function point (FP) are described as measures form which productivity metrics can be computed LOC and FP data are used in two ways during software project estimation:

a)   As an estimation variable to "size" each element of the software.

b)   As baseline metrics collected from past projects and used in conjunction with estimation variable to develop cost and effort projections.

LOC and FP are distinct cost estimation techniques but they have lot of properties in common.

CHECK YOUR PROGRESS:

1.   Decomposition in computer   science,   also   known as ………., is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain.

2.   ……….and …… are measures of cost estimation.

3.   The ………… models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right.

4.   ………   , …… and ……. are the main components of resource pyramid.

5.   What are the four characteristics based on which the software planner specifies resources?

6.   Define component based software Engineering.

5.5 PROJECT PLANNING

Once a project is found to be feasible, software project managers undertake project planning. Project planning is undertaken and completed even before any development activity starts. It involves estimating several characteristics of the project and then planning the project activities based on estimates made. Project planning is taken immediately after the feasibility study phase and before the requirement analysis and specification.

It consists of following essential activities:

a) Estimation- The following project attributes have to be estimated:

Cost – How much it is going to cost to develop the software?

Duration – How long it is going to take to develop the product?

Effort – How much effort would be required to develop the product?

b) Scheduling – After the estimations are made the schedules for manpower and other resources have to be developed.

c) Staffing – Staff organizations and staffing plans have to be made.

d) Risk management – Risk identification, analysis and abatement planning have to be done.

e) Miscellaneous plans – Several other plans such as quality assurance, configuration management plan, etc have to be done.

Project planning requires utmost care and attention since commitment to unrealistic time and resource estimates result in schedule slippage. Schedule delays can cause customer dissatisfaction and adversely affect team morale. It can even cause project failure. For this reason, project planning is considered to be a very important activity.

For large projects it is very difficult to make accurate plans due to the fact that the project parameters, scope of the project, project staff and project managers undertake project planning in stages. In order to deal with these issues the project managers plan the projects in stages that protects the managers to make big commitments too early. This technique of staggered planning is called as Sliding Window Protocol. In this technique starting with an initial plan the project is planned more accurately in successive development stages. After the completion of every phase the project managers can plan each subsequent stages. In addition to the knowledge of the various estimation techniques, past experience is crucial.

CHECK YOUR PROGRESS

7. State True or False

a. Project planning is taken immediately after the feasibility study phase and before the requirement analysis and specification.

b. The past experience is not required to plan the different stages of a project.

---

c.   Cost, Duration and effort are the project attributes which are not required to be estimated.

8.   Define sliding window protocol.

5.6 METRICS FOR PROJECT SIZE ESTIMATION

Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. The accurate estimation of the problem size is fundamental to satisfactory estimation of the other parameters such as effort, time, duration for completing the project and the total cost of developing the software. It is important to understand that the size of the project is very important to estimate to accurately calculate the exact cost of the software development.

Various measures are used in project size estimation. Some of these are:

- Lines of Code

- Number of entities in ER diagram

- Total number of processes in detailed data flow diagram

- Function points

The project size is a measure of the problem complexity in terms of the effort and time required to develop the product. Currently, two metrics (LOC and FP) are popularly being used to estimate size:

5.6.1 Lines of Code (LOC)

The LOC is the simplest among all the metrics available to estimate the project size. As the name suggest, LOC count the total number of lines of source code in a project. The lines used for commenting and the header lines are ignored. Determining the exact line of code at the beginning of the project is difficult. In order to do that one would have to do a systematic guess. The project managers usually divide the problem into modules and each modules into sub modules and so on, until the size of the leaf module can be approximately predicted. In order to predict the LOC count for various leaves past

experience in developing the similar kind of product plays an important role. The units of LOC are:

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

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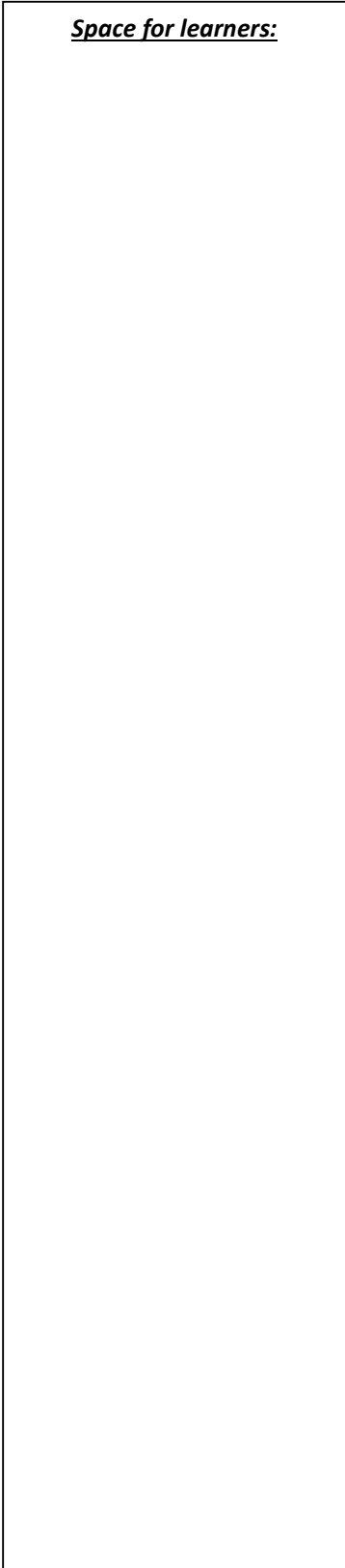
Advantages:

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

Disadvantages:

- Different programming languages contains different number of lines.
- No proper industry standard exists for this technique.
- It is difficult to estimate the size using this technique in early stages of project.
- It also depends on the individual coding style of programmers.
- It only considers the coding effort but other factors like design, test etc. need to be considered in order to calculate the exact effort.
- It is very difficult to calculate the exact size estimate at the beginning of the project and LOC count can only be accurately computed after the code has been fully developed.
- LOC metric measures the lexical complexity of a program and does not address the more important but subtle issues of logical or structural complexities.

5.6.2 Function Point Metric

In this method, the number and type of functions supported by the software are utilized to find FPC (function point count). This metric

overcomes many of the shortcomings of LOC metric. The main advantage of this metric is that it can estimate the size of the project directly from the problem specification. This is completely in contrast with the LOC metric. The main idea behind Function point metric is that the size of the software product mainly depends on the number of different functions or features it supports. A software product with less features is certainly smaller that the product with more features. Each function in a system when invoked reads some input and transforms it into some output. Thus computation of number of input and output to a system gives some idea about the number of functions supported by the system. Albrecht postulated that in addition to the number of basic functions that a software performs, the size is also dependent on the number of files and the number of interfaces. Interfaces refer to the different mechanism that need to be supported for data transfer with other external systems.

The steps in the function point is computed in three stages. The first is to compute the unadjusted function point. In the next step the UFP is refined to reflect the differences in the complexities of the different parameters of the expression for UFP computation. In the final step the FP is computed by further refining UFP for account for the specific characteristics of the project that can influence the development effort.

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

The expression shows the computation of the Unadjusted Function Point as weighted sum of these five problem characteristics. It was validated by Albrecht empirically and was validated through data gathered from many projects. The meaning of different parameters of this expression is as follows:

1)   Number of inputs: Each data item input by the user is counted and it should be distinguished from the user inquiries. The inquiries are the user command that are counted separately. The group of related inputs are considered as a single input and the individual data input items inputted by the user is not counted.

2)   Number of outputs: The output considered refer to reports printed, screen outputs, error messages produced etc. While computing the number of outputs the individual data items within a

report are considered, but a set of related data items is counted as one output.

3)   Number of Inquiries: It is the number of distinctive interactive queries made by the users. These are the use command which require specific action by the system.

4)   Number of files: Each logical file is counted. It implies a group of logically related data. Thus, logical files include data structures and physical files.

5)   Number of Interfaces: The interfaces are used to exchange information with other systems. Examples of such interfaces are files on tapes, disks, communication links with other systems etc.

Advantages:

- It can be easily used in the early stages of project planning.

- It is in depending on the programming language.

- It can be used to compare different projects even if they use different technologies (database, language etc).

Disadvantages:

- It is not good for real time systems and embedded systems.

- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

- The major shortcoming is that it does not take into account the algorithmic complexity of a software. It means it assumes that the design and effort required to develop any two functionalities of the system is same. But normally it is not true.

5.6.3 Number of Entities in ER Diagram

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

Advantages:

- Size estimation can be done during initial stages of planning.

- Number of entities is independent of programming technologies used.

Disadvantages:

- No fixed standards exist. Some entities contribute more project size than others.

- Just like FPA, it is less used in cost estimation model. Hence, it must be converted to LOC.

5.6.4 Total Number of Processes in Detailed Data Flow Diagram

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

Advantages:

- It is independent of programming language.

- Each major processes can be decomposed into smaller processes. This will increase the accuracy of estimation

Disadvantages:

- Studying similar kind of processes to estimate size takes additional time and effort.

- All software projects are not required to construction of DFD.

5.7 PROJECT ESTIMATION TECHNIQUES

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively. Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement. The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae. Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months. The sum of time required to complete all tasks in hours or days is the total time invested to complete the project. This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

➢   Size of software

➢ Software quality

➢ Hardware

➢ Additional software or tools, licenses etc.

➢ Skilled personnel with task-specific skills

➢ Travel involved

➢ Communication

➢ Training and support

There are three broad categories of estimation techniques:

1. Empirical Estimation Techniques

2. Heuristic Technique

3. Analytical Estimation Technique

5.7.1 Empirical Estimation Technique

It is based on the educated guess of the project parameters. While using this technique, prior experience with similar products is helpful. Although, empirical estimation technique is based on common sense but different activities are formalized over years. The empirical cost estimation techniques are based on pure guess work and have over the years formalized to some extent. Two popular methods are:

a) Expert Judgement Technique – It is one of the most widely used technique. In this an expert makes an educated guess of the problem analyzing the problem thoroughly. Usually the expert calculate the cost of different components (modules and subsystems) that would make up the system and then combines the estimates for the individual modules to arrive at the overall estimates. However, this technique is subject to human errors and individual bias. Sometimes the experts overlook some of the factors inadvertently.

b) Delphi Cost Estimation Technique – It tries to overcome the shortcomings of expert judgment technique. It is carried out by a team comprising of a group of experts and coordinators. The coordinator provides the copy of Software Requirement Specification (SRS) document to every member of the team. Estimators analyses the problem domain and after estimating the cost they submit it to the coordinator. The coordinator prepares the summary of the responses of the estimators. The prepared summary is again distributes among

the estimators for further refinement and the process is iterated for several rounds.

5.7.2 Heuristic Technique

The Heuristic technique assumes that the relationship among the different project parameters can be modelled using suitable mathematical expression. Once the independent parameters are known the dependent parameters are calculated using the dependent parameters by substituting the value of basics parameters in the mathematical expression. The COCOMO Model is the heuristic approach of cost estimation technique that we will discuss later.

5.7.3 Analytical Estimation Technique

It derives the required results staring with certain basic assumptions regarding the project. Thus, unlike empirical and heuristic technique the analytical technique does not have certain scientific basis.

CHECK YOUR PROGRESS

13.   ……….. cost estimation techniques tries to overcome the shortcomings of expert judgement approach.

14.   COCOMO is a ……………. Estimation technique.

15.   State true or false.

a.   The heuristic technique makes an educated guess of the project parameters.

b.   The analytical technique of cost estimation have scientific basis.

5.8 COCOMO

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software. In order to classify a product into the identified categories, Bohem requires us to consider not only the characteristics of the product but also those pf development team and development environment. In COCOMO, projects are categorized into three types:

1) **Organic:** A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

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

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

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

5.8.1 Basic COCOMO Model

The basic COCOMO model provides an approximate estimate of the of the project parameters. The following expressions give the basic COCOMO estimation model:

$Effort = a1*(KLOC)^{a2}$ PM

$T_{dev} = b1*(efforts)^{b2}$ Months

where,

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

b) a1, a2, b1, b2 are constants for each group of software products.

c) T_{dev} is the estimated time to develop the software, expressed in months.

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

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

Estimation of development effort:

Organic: Effort = 2.4(KLOC)$^{1.05}$ PM

Semi-detached: Effort = 3.0(KLOC)$^{1.12}$ PM

Embedded: Effort = 3.6(KLOC)$^{1.20}$PM

Estimation of development time:

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

Organic: T_{dev} = 2.5(Effort)$^{0.38}$ Months

Semi-detached: T_{dev} = 2.5(Effort)$^{0.35}$Months

Embedded: T_{dev} = 2.5(Effort)$^{0.32}$Month

The effort required to develop a product increases very rapidly with project size. The size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. The development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type. From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space etc. It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to

complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

The effort and duration values computed by COCOMO are the values for doing the work in the shortest time without unduly increasing the manpower. It is important to note that effort and duration estimations obtained using the COCOMO Model imply that if we try to complete the project in a time shorter than the estimated duration the cost will increase drastically. But if we complete the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost.

5.8.2 Intermediate Model

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

Classification of Cost Drivers and their attributes:

Product – The characteristics of the products that are considered include the inherent complexity of the product, reliability requirements of the product and size of the application database etc.

Hardware - The characteristics of the computer that are considered include the execution speed required, storage space required etc. The other factors include:

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time
- Personnel attributes -
- Analyst capability
- Software engineering capability
- Applications experience

- Virtual machine experience

- Programming language experience

Personal - The characteristics of the development personal that are considered include the experience of personal , programming capability , analysis capability etc.

Development Environment: It captures the development facilities available to the developers. It also includes:

- Use of software tools

- Application of software engineering methods

- Required development schedule

5.8.3 Complete COCOMO Model

The major limitation of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large software systems are made up of smaller subsystems and they have widely different characteristics. For example some have organic type and others may be of semidetached or embedded type. It incorporates all qualities of the standard version with an assessment of the cost driver's effect on each method of the software engineering process. In complete COCOMO the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort. The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

The Six phases of detailed COCOMO are:

1. Planning and requirements

2. System structure

3. Complete structure

4. Module code and test

5. Integration and test

6. Cost Constructive model

To improve the accuracy of their result, the differentiate parameter values of the model can be fine – tuned and validated against an

organization's historical project database to obtain more accurate estimations. Estimations models such COCOMO are not accurate and lack a full of scientific justification. But still software cost estimation model like COCOMO are required for an engineering approach to software project management. Although, these estimates are gross approximations – without such models, one has only subjective judgements to rely on.

5.9 SUMMING UP

- The software resources or the project resources are required for successful development and completion of project. These resources can be capital, people, material, tool or supplies that are helpful to carry out certain task in project. Without these resources it is impossible to complete project.

- In project planning phase identification of resources that are required for completion of project and they will allocated is key element and very important task to do.

- Decomposition Technique uses the concept of "Divide and conquer" method for the cost estimation of a project. By decomposing the projects into major functions and related engineering activities the cost and size estimation can be performed in stepwise fashion.

- Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. The accurate estimation of the problem size is

fundamental to satisfactory estimation of the other parameters such as effort, time, duration for completing the project and the total cost of developing the software.

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

5.10 ANSWERS TO CHECK YOUR PROGRESS

1. Factoring

2. Line of code and Function Point

3. Empirical Estimation model

4. People, Reusable software components and Hardware & Software components.

5. The characteristics based on which the software planner specifies resources are

   a. Description of resource

   b. Resource Availability

   c. Time of Resource when it will be available

   d. Duration of resource availability

6. Component Based Software Engineering - In order to accelerate and bring ease to the software development process the industry prefers to use some ready components of software and this technique is called as Component Based Software Engineering.

7. a. True , b. False , c. False

8. Sliding Window Protocol –

9. Problem Domain

10. Unadjusted Function point

11. Cost

12. a. True , b. False

13. Delphi

14. Cost

15. a. True , b. False

16. a. False , b. True

17. Size , Effort , Duration , Cost

5.11 POSSIBLE QUESTIONS

Short Answer type Questions:

1) Define software decomposition.

2) What are the three main categories of projects in COCOMO Model?

3) What is the difference between Expert judgement and Delphi cost estimation technique?

4) Define LOC and Function Point Metrics.

5) What do you mean by software resources? What are the major types of software resources?

Long Answer type Questions:

1) What is meant by the size of software project? Why does a project manager need to estimate the size of the project?

2) What is project planning and why it is important? What are the different stages of project planning?

3) What are the different categories of software development projects according to COCOMO estimation model?

4) Why is accurate estimation of the effort required for completing a project is difficult? Briefly explain the different effort estimation methods that are available.

5) What are the relative advantage of using either the LOC or the function point metric to measure the size of software product?

6) List the important shortcomings of LOC for use as a software size metric for carrying out project estimations.

7) Explain the Basic COCOMO model briefly. Discuss about its limitations.

8) What is complete COCOMO Model? Explain briefly.

9) What do you mean by Intermediate COCOMO? What are the major cost drivers that are considered in Intermediate COCOMO?

10) Explain why adding more man power to an already late project makes it later.

5.12 REFERENCES AND SUGGESTED READINGS

- "Fundamentals of Software Engineering", Rajib Mall, Prentice-Hall of India.

- "An Integrated Approach to Software Engineering", Pankaj Jalote , Narosa Publishing House

- http://www.tutorialspoint.com

- http://www.geeksforgeeks.com

UNIT 6: SOFTWARE DESIGN I

Unit Structure:

6.1 INTRODUCTION

Software design is an important phase in software engineering, in which a blueprint is designed to serve as a base for constructing the software system. The design process comprises a set of principles, concepts and practices, which permit the software designer to model the system or product which is to be built. It is a process to transform user requirements into some suitable form, which helps the programmer in writing software coding and implementation. The design phase in Software Development Life Cycle (SDLC) plays a crucial role in obtaining a quality software product. Here, the system is designed to satisfy the identified requirements in the previous phases and then they are transformed into a System design document that

accurately describes the system design. This chapter will focus on the design concepts and elements that are required to develop a software design model.

6.2 UNIT OBJECTIVES

After completion of this unit, you will be able to learn --

- The standard definition of software design.

- The factors behind the qualities of a good design.

- The constraints behind the software design.

- The key design concepts like abstraction, modularity, information hiding, functional independence, cohesion and coupling.

6.3 DEFINITION OF SOFTWARE DESIGN

The design activity begins when the requirements document for the software to be developed becomes ready. The design of a software is essentially a plan or blueprint to serve as a foundation for constructing the software system. According to IEEE, software design can be defined as 'both a process of defining, the architecture, components, interfaces, and other characteristics of a system or component and the result of that process.' Viewed as a process, software design can be considered an activity within the software development life cycle, where software requirements are analyzed in order to produce a description of the internal structure and organization of the system that will serve as the basis for its construction. More precisely, a software design must describe the architecture of the system, that is, how the system is decomposed and organized into components and must describe the interfaces between these components. It must also describe these components into a level of detail suitable for allowing their construction.

6.4 QUALITIES OF A GOOD SOFTWARE DESIGN

For developing a good quality software, the software design must also be of good quality. Now, the matter of concern is how the quality of a good software design is measured? This is done by observing certain factors in software design. The definition of a good software design may vary based on the exact application being designed because the criteria used to judge a design solution depend on the application being designed. However, some deserving factors which are associated with a good software design for general applications may be listed in the following way:

- Correctness

- Understandability

- Efficiency

- Maintainability

Now, let us define each of them in detail.

- Correctness

The design of any software is assessed for its correctness first. The evaluators check the software for every kind of input and action, and observe the results that the software produces according to the proposed design. If the outputs are correct for every input, the design is accepted and considered that the software produced according to this design will function correctly.

- Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. The reason is that there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. Therefore, the developers find easy to implement and build the same software that is represented in the design if the design is easy and self- explanatory.

- Efficiency

The software design must be efficient. The efficiency of the software can be evaluated from the design phase itself, because if the design describing the software is not found efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

- Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time updating and maintenance. So, the design of the software must also be able to handle such changes. It should not be the case that after making some modifications, the other features of the software start nonfunctioning. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

6.5 DESIGN CONSTRAINTS

A constraint is anything that slows a system down or prevents it from achieving its goal. Design constraints are some challenges that force people for considering more methodical analysis of design and their problems. Because resources are not inexhaustible and criteria must be met. Software designers are to be more strategic about the processes they employ and energies they spend. In order to address design constraints, a straightforward approach is required to categorize the type of constraints (e.g., hardware, software, procedure, algorithm), identify the specific constraints for each category, and capture them as system requirements.

Types of constraint

The following kind of constraints might encounter in the theory of design constraints.

➢ Policy

Policy constraints are those caused by the company procedures and policies. A policy constraint in the process of developing a software

might relate to security / compliance requirements. On the other hand, it could be an issue with interchanging the code between team members.

> ➤ Equipment

Equipment constraints refer to delays caused by faulty, slow, or outdated equipment or a lack of sufficient space. In software development, this might be faulty keyboards or slow computers. It might also be a lack of devices through which cross-platform tests are to be run.

> ➤ People

A people constraint is a common challenge caused by the number of people involved in a project. Often, people constraints are caused by unavailability of skilled people for a project. On the other hand, in software development, having too many skilled people on a project can also cause a people constraint.

> ➤ Paradigm

A paradigm constraint is a constraint caused by opinions. The view like, for example, 'lines of code' is considered a good metric for productivity, when the opposite can often be true.

> ➤ Market

A market constraint is related to the constraint which lies in delivering a software to consumers. In software, this would look like overengineering and feature creep.

6.6 FUNDAMENTAL DESIGN CONCEPTS

Every software process is characterized by basic concepts along with certain practices or methods. Methods represent the manner through which the concepts are applied. As new technology replaces older technology, many changes occur in the methods that are used to apply the concepts for the development of software. However, the fundamental concepts underlying the software design process remain the same, some of which are described in the following.

6.6.1 Abstraction

As a powerful design tool, abstraction is meant for allowing software designers to define components at an abstract level by ignoring the implementation details of the components. IEEE defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.' The concept of abstraction can be visualized in two ways: as a process and as an entity. As a process, it refers to a mechanism of hiding irrelevant details and representing only the essential features of an item and thereby one can concentrate on important things. In terms of an entity, it refers to a model or view of an item.

Every step in the software process is realized through various levels of abstraction. At the highest level, a framework of the solution to the problem is presented whereas at the lower levels, the detail solution to the problem is outlined. For example, in the requirements analysis phase, a solution to the problem is presented using the language of problem environment and as we proceed through the software process, the abstraction level reduces and at the lowest level, source code of the software is produced.

There are three commonly used abstraction mechanisms in software design, namely, functional abstraction, data abstraction and control abstraction. All these mechanisms allow us to control the complexity of the design process by proceeding from the abstract design model to concrete design model in a systematic manner.

- Functional abstraction: This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'. Within these groups, there exist routines which may be visible or hidden. Visible routines can be used within the containing groups as well as within other groups, whereas hidden routines are hidden from other groups. These can be used within the containing group only.

- Data abstraction: This process concentrates on specifying data that describes a data object. For example, the data object *window* encompasses a set of attributes (window type, window dimension) that describe the window object clearly. In this

abstraction mechanism, representation and manipulation details are ignored.

- Control abstraction: This abstraction provides the programmer the ability to hide procedural data. It refers to the software part of abstraction wherein the program is simplified and unnecessary execution details are removed. For example, if and while statements in programming languages (like C and C++) are abstractions of machine code implementations, which involve conditional instructions. In the architectural design level, this abstraction mechanism permits specifications of sequential subprogram and exception handling without the concern for exact details of implementation.

6.6.2 Information Hiding

Information hiding states that each module should hide a design decision from the rest of the modules. In particular, the designer should choose to hide within a module an aspect of the system that is likely to change as the program evolves. Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules. They pass only that much information to each other, which is required to accomplish the software functions. The way of hiding unnecessary details is referred to as information hiding. IEEE defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus, each module is a 'black box' to the other modules in the system.'

6.6.3 Modularity

Modularity is attained by splitting the software into uniquely named and addressable components, which are also known as modules. A modular system can be characterized by functional partitioning into these type of discrete scalable and reusable modules, rigorous use of well-defined modular interfaces and making use of industry standards for interfaces. A complex system (large program) is partitioned into a

set of discrete modules in such a way that each module can be developed independent of other modules. After designing all the modules, they are combined together to meet the software requirements as specified in SRS document. It is to be noted that larger the number of modules a system is divided into, greater will be the effort required to integrate the modules. Figure 6.1 depicts the modularity concepts of a software system. This graphical design of modularity clearly states that a program can be intellectually manageable if its activities are modularized.   The desirable characteristics of modular design can be listed as:

- Each module is well defined system that can be used with other applications.

- Each module has a single specific job.

- Modules can be separately compiled and stored in a library.

- A module can employ other modules.

- Modules are easier to use than to build.

- Each module is simpler from outside than inside.

Thus, modularity enhances the clarity of design which in turn simplifies coding, testing, debugging, documenting and maintenance of a software system. Modular design usually follows the rules of 'divide and conquer' problem-solving strategy because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- Smaller components are easier to maintain.

- Program can be divided based on functional aspects.

- Desired level of abstraction can be brought in the program.

- Components with high cohesion can be re-used again.

- Concurrent execution can be made possible.

- Desired from security aspect.
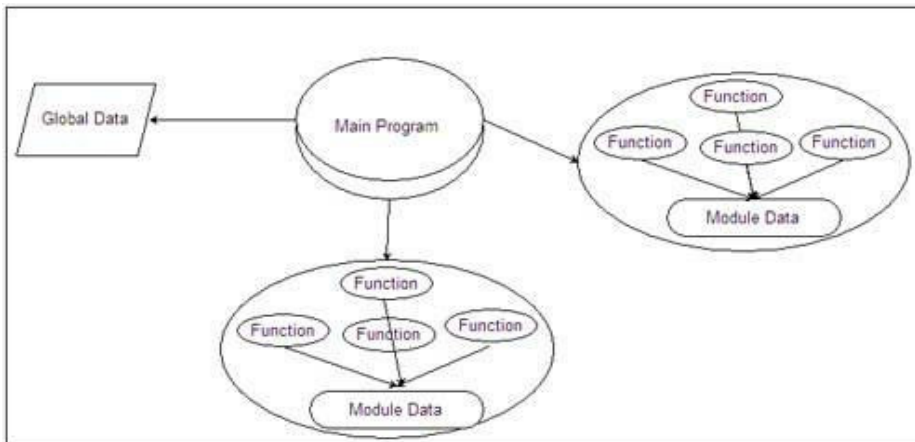
Figure 6.1 Modularity concepts of a software system

With the introduction of modular design, complexity of software design has considerably reduced. It facilitates the change in the program which in turn encourages parallel development of systems. Modularizing a design helps to plan the development in a more effective manner, accommodates any changes easily and also helps to conduct testing and debugging effectively and efficiently. Also, conducting maintenance work without adversely affecting the functioning of the software is another attractive advantage of modularity.

6.7 SOFTWARE DESIGN LEVELS

Software design yields three levels of results:

- Architectural Design - The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

- High-level Design – It is related to identification of different modules and the their control relationships and the definition of the interfaces among these modules. The outcome of the high-level design is called software architecture or the program structure. Many different types of notations have been used to represent a high-level design. High-level design focuses on how the system along with all of its components can be implemented

in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

- Detailed Design- Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. During detailed design, the data structure and the algorithms of the different modules are designed. Moreover, it defines logical structure of each module and their interfaces to communicate with other modules.

6.8 MODULARIZATION CRITERIA

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As explained in earlier section, it is clear that the concept of modularity can reduce the complexity by breaking a system into varying degrees of interdependence and independence across, and hide the complexity of each part behind an abstraction and interface. A module is a set of instructions put together in order to achieve some specific tasks. They are though, considered as single entity but may refer to each other to work together. There are some procedures by which the quality of modular design and interaction among them can be measured. These measures are called coupling and cohesion. Conceptually, coupling and cohesion are two qualitative criteria of functional independence. Functional independence is the refined form of the design concepts of modularity, abstraction and information hiding as described in earlier sections. It is achieved by designing a module in such a way that it independently performs given set of functions without interacting with other parts of the system.

6.8.1 Coupling

Coupling is a measure that defines the level of inter-dependability among the 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. Various factors such as type of data that pass

across the interface, interface complexity, number of interfaces per module, etc. impact on the strength of coupling between two modules.

There are five levels of coupling, namely -

- Data coupling: Data coupling is the one when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components. This way, communication between two modules is achieved. In data coupling, the components are independent to each other and communicating through data. Data coupling is considered the best coupling among all.

- Stamp coupling: When multiple modules share common data structure and work on different part of it, it is called stamp coupling. In this coupling, the complete data structure is passed from one module to another module.

- Control coupling: Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution. In this coupling, modules communicate by passing control information. It is considered bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality.

- Common coupling: When multiple modules have read and write access to some global data, it is called common or global coupling. The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change.

- Content coupling: When a module can directly access or modify or refer to the content of another module, it is called content coupling. Here, control flow can also be passed from one module to the other module. This is the worst form of coupling and should be avoided.

6.8.2 Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The tighter the elements are bound to each other, the higher is the cohesion of a module. The greater the cohesion, the better is the program design. Low coupling results in high cohesion and vice versa. Hence, designers should maintain a high-level cohesion while designing a module.

There are seven types of cohesion, namely –

- Functional cohesion - It is considered to be the highest degree of cohesion, and it is highly expected. Elements within modules in this cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

- Sequential cohesion - When elements within a module are grouped and involved in activities in such a way that the output of one element serves as input to another and so on, it is called sequential cohesion.

- Communicational cohesion - When elements of different modules are grouped together, to perform different functions and work on same data (information), it is called communicational cohesion.

- Logical cohesion - When logically categorized elements are put together into a module, it is called logical cohesion.

- Temporal Cohesion - When elements within modules are organized in such a way that they are processed at a similar point in time, it is called temporal cohesion.

- Procedural cohesion – In this cohesion, elements of within modules are involved in different and possibly unrelated activities which are executed sequentially in order to perform a task.

- Co-incidental cohesion - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

From the above discussion on coupling and cohesion, it can be summarized the difference between them in the following way.

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
It conceptualizes the relationships between modules.	It shows the relationship within the module.
It shows the relative independence between the modules.	It defines the relative functional strength of a module.
In coupling, modules are linked to the other modules by passing either data structure or control information.	In cohesion, the module focuses on a single thing.

CHECK YOUR PROGRESS

A. Choose the correct options from the following:

1. _____ refers to a powerful design tool, which allows software designers to consider components at an abstract level ignoring the implementation details of the components.

    A. Information hiding

    B. Functional decomposition

    C. Abstraction

    D. None of these

2. Software design yields _____ levels of results.
    A. 2
    B. 3
    C. 4
    D. 5

3. Which of the following is not an advantage of modularization?

A. Smaller components are easier to maintain

B. Concurrent execution can be made possible

C. Program cannot be divided based on functional aspects

D. Desired level of abstraction can be brought in the program

4. Which of the following defines the degree of intra-dependability within elements of a module?

A. Cohesion

B. Coupling

C. Design Verification

D. None of the above

5. When multiple modules share common data structure and work on different part of it, it is called _____.

A. Common coupling

B. Share coupling

C. Data coupling

D. Stamp coupling

6. In Design phase, which is the primary area of concern?

A. Architecture

B. Data

C. Interface

D. All of the above

7. Which of the following is the best type of module cohesion?

A. Functional Cohesion

B. Temporal Cohesion

C. Co-incidental Cohesion

D. Sequential Cohesion

8. Which of the following is the worst type of module coupling?

A. Control Coupling

B. Stamp Coupling

C. Data Coupling

D. Content Coupling

9. How many types of cohesion are there in software design?

A. 5

B. 6

C. 7

D. 8

10. Which design identifies the software as a system with many components interacting with each other?

A. High-level design

B. Architectural Design

C. Detailed design

D. Both A & C

B. State True or False:

1. Modules should be specified and designed in such a way that the data structures and processing details of one module are accessible to other modules.

2. Stepwise refinement is a top-down design strategy used for decomposing a system from high-level abstraction to more detailed form of abstraction.

3. Information hiding makes program maintenance easier by hiding data and procedure from the unaffected parts of a system.

4. Since modularity is an important design goal, it is not possible to have too many modules in a proposed design.

5. Keeping low cohesion and high coupling is good practice for a software designer.

6.9 SUMMING UP

o Software design is a stage in software development life cycle in which a blueprint is developed to serve as base for constructing the software system.

o There are various software design concepts which lay the foundation for the software design process.

o Abstraction refers to the process which allows software designers to consider components at an abstract level, ignoring the implementation details of the components.

o Modularity is the process of decomposing the software into uniquely named and addressable components called modules.

- o Modules should be designed in such a way that the data structures and processing details of one module are not accessible to other modules. Only the required information is passed among
- the modules to accomplish the software functions. This is called information hiding.
- o Functional independence refers to the use of parameterized subprograms or groups and within these groups there exist routines which may be accessible or hidden.
- o Data abstraction involves how to specify data that describes a data object. Again, control abstraction hides the procedural data stating only the desired effect.
- o Functional independence is the refined form of the design concepts of modularity, information hiding and abstraction. Each module is developed independently so that it uniquely performs given set of functions without interacting with other sections of the system.
- o Coupling and Cohesion are two qualitative criteria for measuring functional independence.
- o Coupling measures the degree of interdependence among the modules whereas cohesion is the measurement of the relative functional strength of a module.

6.10 ANSWERS TO CHECK YOUR PROGRESS

Answers to A:

1. C	2. B	3. C	4. A	5. A
6. D	7.A	8. D	9. C	10.B

Answers to B:

1. True	2. True	3. True	4. False
5. False			

6.11 POSSIBLE QUESTIONS

- What are the common features a good software design must possess?

- How do you explain the terms cohesion and coupling in the context of software design?

- Discuss the different types of cohesion that a module might exist with proper example.

- Enumerate the various types of coupling that might exist between two modules with example of each.

- Differentiate coupling with cohesion.

- What do you understand by the term functional independence? What are the advantages of functional independence?

UNIT 7: SOFTWARE DESIGN II

Unit Structure:

7.1 INTRODUCTION

As discussed in Unit 6, Software design can be defined as a plan for converting a specification into executable code i.e., problem description turned into problem solution. Software design involves two major categories of design methodologies: structural design and algorithmic design. Structural design further involves many levels of system decomposition followed by algorithmic design. All software systems are called information-processing systems because they accept data as input, process the input and finally provide the data in result form. Therefore, software design must concentrate on three basic issues:

- definition of the data to be held by the system;

- definition of the process by which inputs are manipulated;

- formulation of states that the system can assume and what transformations are required between states.

7.2 UNIT OBJECTIVES

After completion of this unit, you will be able to learn --

1. Various methods of software design such as function oriented design and object-oriented design.

2. Several kinds of design notations including Flow chart, Data flow diagram, HIPO diagram, Decision table, Pseudocode.

7.3 SOFTWARE DESIGN METHOD

A design method provides a way of indicating how to create a design. It needs to impart the procedures for verifying that the design is correct. Software design takes the user requirements as challenges and tries to find out optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution. There are mainly three variants of software design methods which are described in the following.

- **Structured Design**

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. The benefit of structured design is that it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design follows 'divide and conquer' technique where a problem is decomposed into several small problems and each small problem is individually solved until the whole problem is solved. The small pieces of problem are solved by means of solution modules which have been well organized in order to achieve precise solution. These

modules are arranged in hierarchy where top level module communicates with descendant modules.

A good structured design follows some rules for communication among multiple modules by the virtue of cohesion and coupling as discussed in previous unit. Cohesion refers to grouping of all functionally related elements within a module whereas Coupling measures the inter dependency among different modules. A good structured design has high cohesion and low coupling arrangements.

- **Function Oriented Design**

In function-oriented design, the system is considered a group of many smaller sub-systems known as functions and hence, system is known as top view of all functions. These functions are capable of performing significant task in the system.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used. This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation. These functional modules can share information among themselves by means of information passing.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Design Process:

➢ The whole system is seen as how data flows in the system by means of data flow diagram.

➢ Data flow diagram depicts how functions changes data and state of entire system.

➢ The entire system is logically broken down into smaller units known as processes on the basis of their operation in the system.

➢ Each function is further described at large by decomposing the same into multiple sub functions.

- **Object Oriented Design**

Object oriented design (OOD) works around the entities and their characteristics instead of functions involved in the software system. This design strategies mainly concentrates on entities and its characteristics. The whole concept of software solution revolves around the engaged entities. Some important concepts behind Object Oriented Design are:

- Objects - All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it with specified values and those values are processed by some methods to perform respective operation.

- Classes - A class is a generalized description of an object (or a class can be called a template for defining object(s)). An object, on the other hand, is a runtime instance of a class. All the attributes associated with an object along with methods, which defines the functionality of the object are declared within a class.

   In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

- Encapsulation - In OOD, the binding of attributes (data variables) and methods (operation on the data) in a single platform is referred to as encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.

- Inheritance - OOD permits similar classes to stack up in hierarchical manner where the lower or sub-classes can inherit, re-use and implement allowed variables and methods from their immediate parent class (or classes). This property of OOD is known as inheritance. This makes it easier to define specific class or classes known as base class and to construct generalized classes from specific ones.

- Polymorphism - OOD languages provide a mechanism to define more than one method (vary in arguments) having same name performing multiple tasks. This is called polymorphism, which allows a single interface performing tasks for different types.

Depending upon how the function is invoked, respective portion of the code gets executed.

CHECK YOUR PROGRESS

1. What is design method? Name different types of design methods.
2. Differentiate between function-oriented design and object oriented design.

7.4 SOFTWARE DESIGN NOTATION

It can be stated that software analysis and design stage comprise of the activities which help in transforming of requirement specification document into implementation. Requirement specifications specify all functional and non-functional necessities for the software to be developed. These requirement specifications come in the shape of human readable and understandable formats to which a computer has nothing to do. Design notations refers to some techniques which are used to represent a system in software design. These notations help software designers to get an overview of various aspects of software design like modules, abstraction, information hiding, concurrency, etc. in a comprehensive manner.

A design notation in well-formed helps to clarify the relationships and interactions among various modules that exist in the design, while a poor design form generally complicates the design process. It is to be noted that software design notations may be in the form of graphical, textual, or symbolic. Various design notations which are widely used by software designers as analysis and design tools include Flow Charts, Structure Charts, Data Flow Diagram, HIPO diagram, Pseudocodes, Decision Table, etc.

7.4.1 Flow Charts

A flowchart is a design representation in graphical form which shows the sequence of operations to be carried out to solve a given problem. It helps to determine the major elements of a process by creating boundaries between the end of one process and beginning of another process. The logic of a problem can be clearly understood by the programmer from the flow chart of the problem. It makes use of set of symbols which are

---

connected among them to indicate the flow of information and processing. Frank Gilberth introduced flowcharts in 1921, and they were called "Process Flow Charts" at the beginning.

Symbols used in the Flowcharts:

Some standard symbols and rules are prescribed by American National Standard Institute (ANSI) for drawing flowcharts. Table 1 presents some standard symbols which are frequently used in flowcharts.

Table 1. Some standard symbols and their purposes used in flowcharts

Symbol	Symbol Name	Purpose
	Start/Stop	It is used to denote start and end of a logic/program.
	Process	It specifies how to process mathematical operations.
	Input/ Output	Used for accepting inputs for the program and producing outputs by the program.
	Decision box	It stands for decision making statements in a program, where any decision results either Yes or No.
	Flow line	It represents the flow of the sequence and direction of a process.
	On-page Connector	It connects two or more parts of a flowchart, which are on the same page.
	Off-page Connector	It is used to connect two parts of a flowchart which are spread over different pages.

Benefits of Flowcharts:

The sequential steps in an algorithm are pictorially represented by the flowchart and therefore, flowchart is considered a step next to an algorithm. It helps to clarify the actions to be taken. It also allows to improve the currently working things. The misplaced steps or

unnecessarily included steps are clearly come into picture for the software designers with the help of flowchart. Some benefits of using flowchart in problem solving are described in the following.

➢ Logic understanding

Since the flow chart pictorially represents the actions to be performed, the logic used for solving the problem can be understood easily. The flow chart symbols used to perform various functions denote the actions and their flow. Thus, the control flow in a program can be easily visualized with the help of flowchart.

➢ Communication

A common understanding about the process is established among the members of designer team with the help of flow chart. It is a better way of communicating the logic of a system to all concerned. The flowchart makes the communication easier to all the involved people as compared to actual program code.

➢ Effective analysis

Although it is the duty of the programmer to analyse the problem, but it can be handed over to other persons who may or may not be aware of the programming techniques because of flowchart which gives broad idea about the logic. The testing and analysis of the logic is performed in an unbiased manner by these people. Thus, the analysis of the problem becomes effective and easier because of flowchart.

➢ Useful in coding

When the flowchart becomes ready, the start and end of a problem solution become fixed along with all the necessary sequential steps. It acts as a guide to the programmer in planning the coding process effectively. Thus, the flowcharts allow the programmer to develop error free programs in high-level languages at a faster rate.

➢ Proper testing and debugging

The errors in the program can be easily detected by going through the flowchart. The logic used for solving the problem is exactly known to the developer. The developer can test a program by fetching various data and thereby, flow chart allows the testing of program in every contingency.

➢ Appropriate documentation

Flowcharts can serve as good documentation tool for beginners which may not have any programming idea. They can understand what the program actually does and how to use the program with the help of documentation.

To understand the basic concept of flowchart, few examples are discussed in the following.

Example 1. A flow chart is presented in the figure 7.1 which describes to find out the largest of the given three numbers stored in A, B & C.
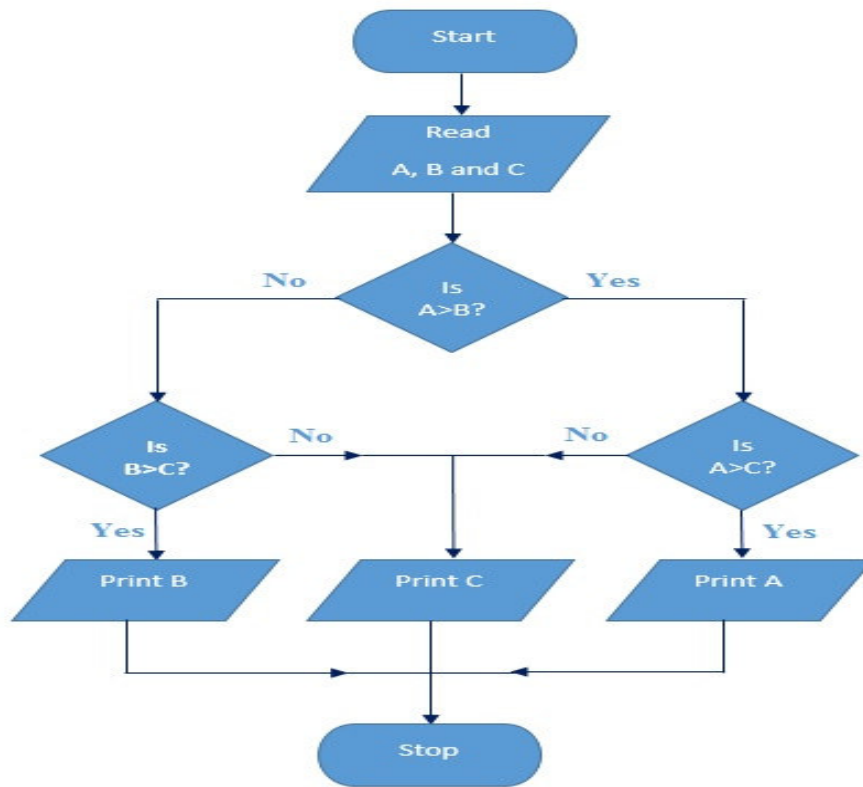

Figure 7.1 Flow chart for finding out the greatest number among three integers

Example 2. A flow chart to generate Fibonacci series upto n which is presented in figure 7.2.
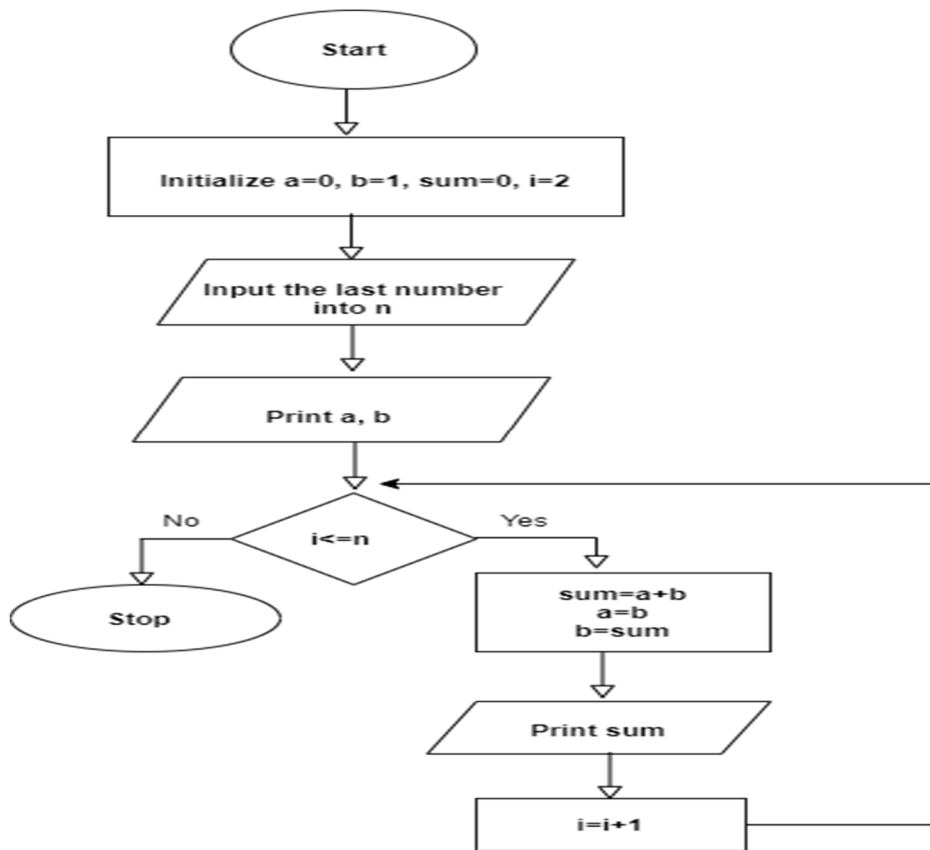
Figure 7.2 Flow chart to generate Fibonacci series upto n.

7.4.2 Data Flow Diagrams

A Data flow diagram (DFD) is a graphical representation of flow of data in an information system. It can depict incoming data flow, outgoing data flow and stored data throughout the system. The objective of DFD (DFD is also known as bubble chart) is to provide an overview of the transformations in the input data within the system in order to produce the output. A DFD is defined by IEEE as 'a diagram that depicts data sources, data sinks, data storage and processes performed on data as nodes and logical flow of data as links between the nodes.' Since graphical representations are easier to interpret compared to the technical descriptions, the non-technical users can also able to understand the system details easily and clearly.

There is a prominent difference between DFD and Flowchart. The flowchart presents flow of control in program modules whereas DFDs

depict flow of data in the system at various levels. DFD does not have any control or branch elements.

Data Flow Diagrams may be either Logical or Physical.

- Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system. It presents the theoretical process of moving information through a system, like where the data comes from, where it goes, how it changes, and where it ends up.

- 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.

DFD notations:

To construct DFD for a system, a set of symbols are used which are standardized notations like rectangle, circle, arrows, etc. There are four common methods of notation used in DFDs: Yourdon & De Marco, Gene & Sarson, SSADM and Unified. All methods use the same labels with different shapes to represent the four main elements of a DFD - external entity, process, data store, and data flow. Figure 7.3 presents these elements. The notations belonging to Yourdon & De Marco method are widely used to construct DFDs. Unified modelling notations are used for defining a system in object-oriented design.
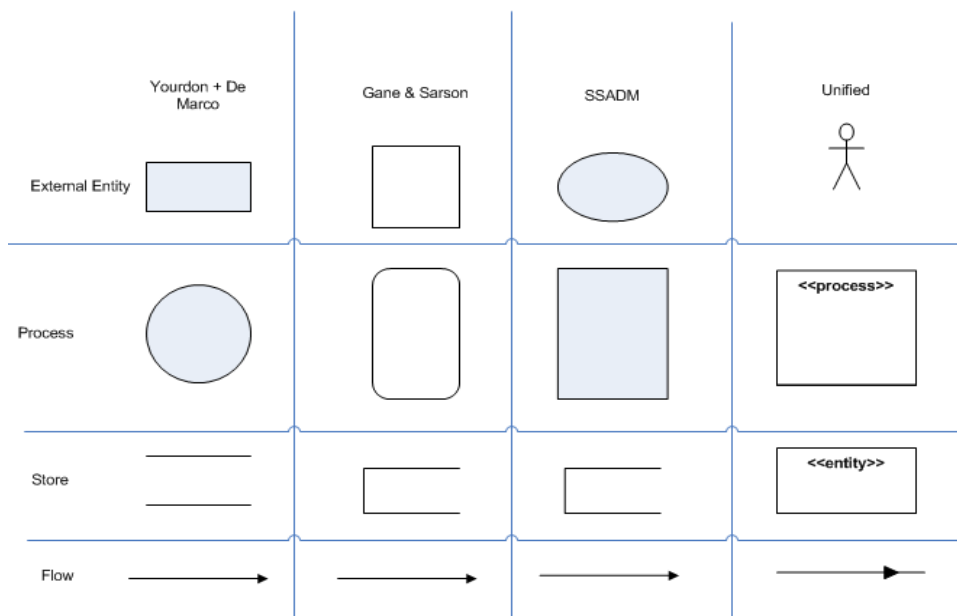
Figure 7.3 Four fundamental notations used in DFDs

External Entity

An external entity, which are also known as terminators, sources, sinks, or actors, represents an external user that sends or receives data to and from the system. They are used either the sources or destinations of information and accordingly, they are usually placed on the diagram's edges. Entities are represented by rectangles with specific names.

Process

Process refers to an operation that manipulates the data and its flow by taking incoming data, changing it, and producing an output with it. A process can do this by performing computations and using logic to sort the data, or change its flow of direction. Activities and action taken on the data by processes are represented by Circles.

Data Store

Data stores hold information for later use, like a file of documents from which necessary data can be retrieved for processing. Data inputs flow from the external entity to a data store through a process while data outputs flow out of a data store to the entity through a process. It is to be noted that data flow can never be possible in between entity and the data store. There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

Data Flow

Data flow is the path by which the system's information passes from external entities through processes and data stores. With arrows along with brief labels, the DFD presents the direction of the data flow. In general, data flow is not shown in between two processes. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Certain standard guidelines are to be followed while creating a DFD. These include the following:

a) DFD notations should be given meaningful names with proper parts of speech. For example, a process name should be a verb whereas

nouns should be used for naming external entities, data store and data flow.

b) Each process should be numbered uniquely and numbering should be consistent.

c) Abbreviations should be avoided in DFD notations.

d) Looping concept should not be used in DFD.

e) A DFD refinement should be continued in a series of levels of DFDs until each process performs a simple operation.

f) The data store should not be depicted in Context diagram or Level 0 DFD, but all the data stores required for a system must be included in next level DFDs.

g) Each process should have at least one input and one output.

h) Each data store should have at least one data flow in and data flow out.

Levels of Data Flow Diagrams

There are various levels of DFD ranging from simple overviews to complex, granular representations of a system or process with deeper levels, starting with level 0. These provide details about the input, processes and outputs of a system. The most common and intuitive DFDs are level 0 DFDs, which are also known as context diagrams.

Level 0 DFD or Context Diagram

This DFD level focuses on high-level system processes or functions and the data sources that flow to or from them. All the external entities must be shown in Level 0 data flow diagram. They are designed to be simple, straightforward overviews of a process or system. It is to be mentioned that no data store should be included in context diagram.

Level 1 DFD

Level 1 DFD elaborates level 0 DFD by splitting the system's single process into more detailed form showing all the broad level functions of the system. It depicts basic modules in the system and flow of data among various modules. It also includes all the data stores required for the system. However, external entities may not be a part of level 1 DFD or other deeper level DFDs.

Level 2 DFD

The level 2 DFDs present more elaboration by breaking down each level 1 process into granular subprocesses.

Level 3 DFD

Level 3 and higher-numbered DFDs are usually not to be defined. This is mainly due to the requirements of large amount of detail which increases the complexity of the system. Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Let us take an example to explain DFD for a system "Online Shopping System". Online Shopping System is an e-commerce application which allows customer to buy goods or services from a seller over the Internet. This system is handled by two types of users: buyer(customer) and seller(Administrator). Hence, two external entities are involved in this system. The context diagram of this system is presented in the figure 7.4.

Figure 7.4 Context diagram of Online Shopping System

Further, the process(system) in context diagram is decomposed into various subprocesses which are presented in Level 1 DFD. Figure 7.5 depicts the Level 1 DFD.
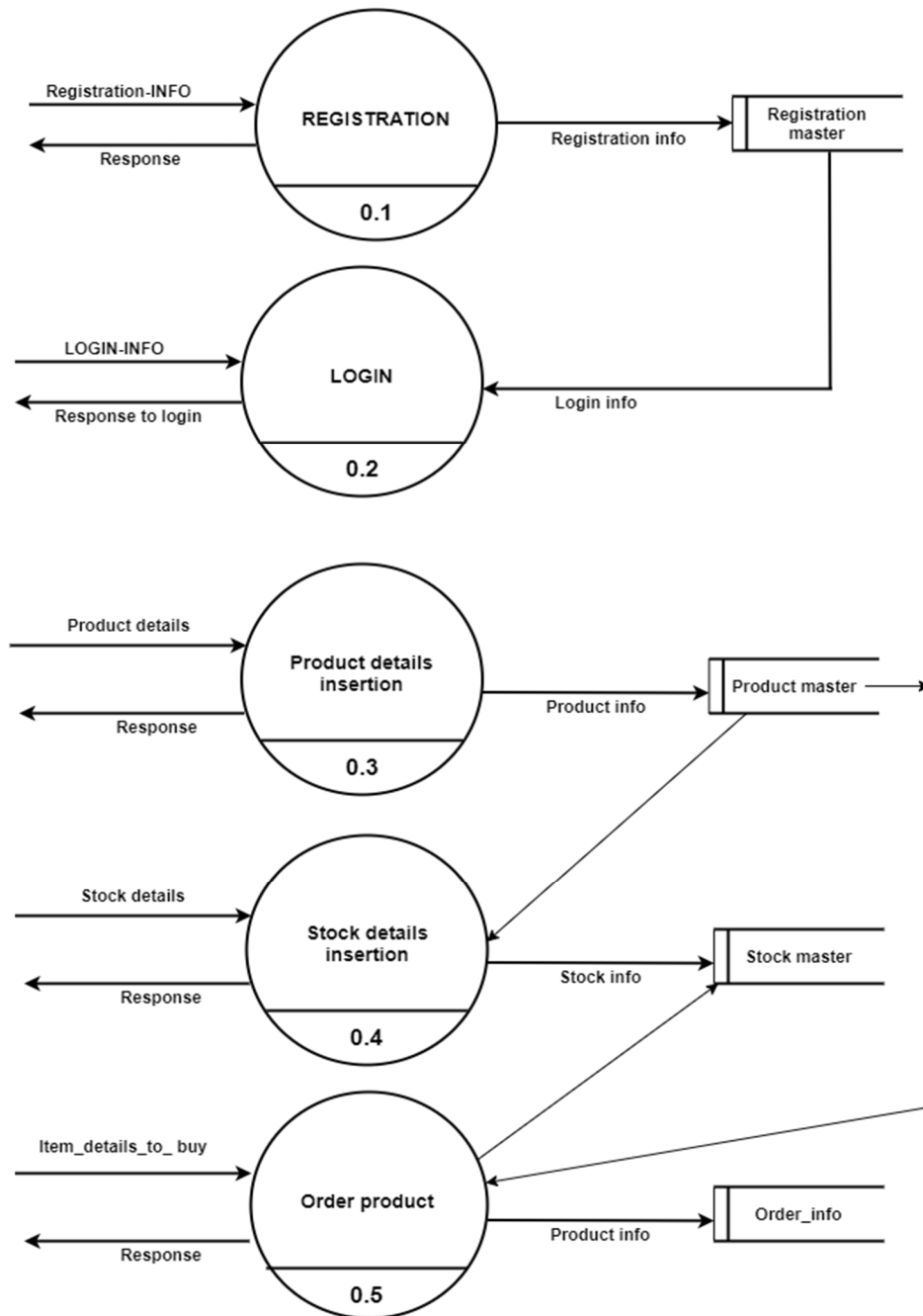
Figure 7.5 Level 1 DFD for Online Shopping system

7.4.3 Pseudo-Code

Pseudocode is a "text-based" detail (algorithmic) design tool. It describes a piece of code or an algorithm. As the name suggests, it does not refer to the actual code. The term is widely used in algorithm-based fields. It is written closer to programming language and therefore it is considered as augmented programming language, full of comments and descriptions. Pseudo code is an implementation of the algorithm in English. It has no syntax like any other programming language, hence it cannot be compiled or interpreted by a computer.

How to write Pseudo code?

Arrange the sequence of tasks before writing the pseudo code.

1. Start with the statement which establishes the main goal or the aim.

2. Use if-else like statement, for, and while loops wherever required and indent the statements.

3. Make use of appropriate conventions. If the programmer goes through a pseudo code, the naming must be simple and distinct.

4. Use appropriate sentence casings such as Proper Case for functions or methods, Upper case for constants and Lower case of variables.

5. Elaborate on everything that is going to happen in the actual code.

6. Check whether all the sections of a pseudo-code are complete, finite and clear.

7. It is not advisable to write the code in a complete programming manner.

8. Avoid variable declaration in Pseudo code.

Why use Pseudo code?

The pseudo code improves the readability of any approach. It is the best approaches to start the implementation of an algorithm. This kind of practice helps in bridging the potential gaps between the program, algorithm, or flowchart. It also acts as a rough document.

The rules of Pseudocode are reasonably straightforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch. Examples below will illustrate this notion. Pseudo code

contains more programming details than other design notations. It provides a method to perform the task, as if a computer is executing the code. Few examples of Pseudocode are mentioned below.

- ✓ To print the result of a student as "passed" if he/she obtains average marks equal to or more than 60 otherwise "failed".

  Input the student's grade

  If student's grade is greater than or equal to 60

  Print "passed"

  else

  Print "failed"

- ✓ To compute the average grade mark of 10 students of a class.

  Set total to zero

  Set grade counter to one

  While grade counter is less than or equal to ten

  Input the next grade

  Add the grade into the total

  Set the class average to the total divided by ten

  Print the class average.

- ✓ Finding the sum of n natural numbers.

  Declare variables n, i and sum as integer;

  Read number n ;

  Initialize i to 1

  for i upto n increment i by 1

  {

  sum=sum+i;

  }

  Print sum;

✓ Program to print Fibonacci up to n numbers.

    void function Fibonacci

    Get value of n;

    Set value of a to 1;

    Set value of b to 1;

    Initialize i to 0

    for (i=0; i< n; i++)

    {

    if a greater than b

    {

    Increase b by a;

    Print b;

    }

    Else if b greater than a

    {

    Increase a by b;

    print a;

    }

    }

7.4.4 Structure Charts

Structure chart is a chart derived from Data Flow Diagram (DFD). It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules describing functions and sub-functions of each module of the system to a greater detail than DFD.

Structure charts are also the graphical representation of the high-level design which presents the hierarchical structure of modules. At each layer, a specific task is performed. As a design notation, structure charts

show both control and data flow between modules. Here are the symbols used in construction of structure charts.

- Module - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.
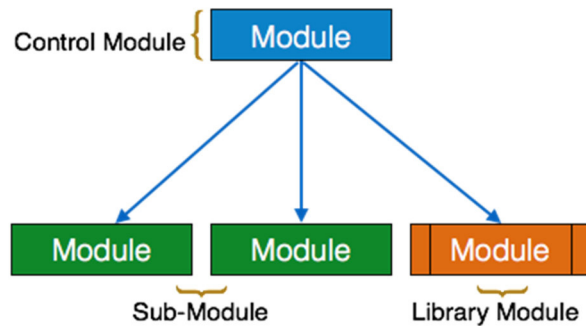
- Condition - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.
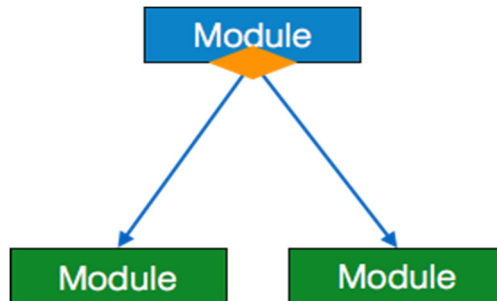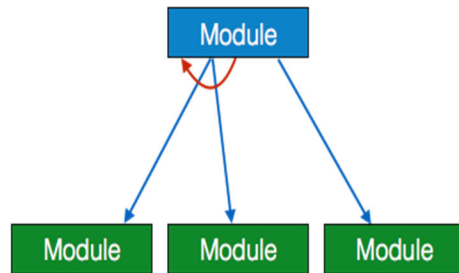

- Jump - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.
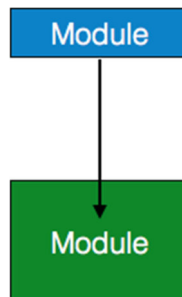
- Loop - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.

- Data flow - A directed arrow with empty circle at the end represents data flow.
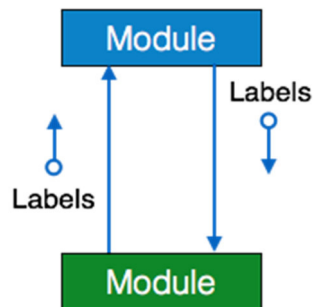
- Control flow - A directed arrow with filled circle at the end represents control flow.

-
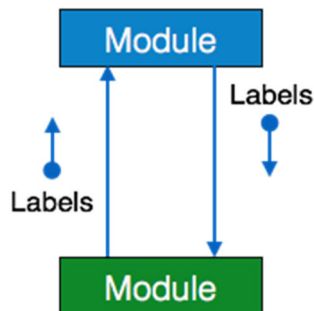
Developing a Structure Chart:

To build a structure chart, the steps given below are to be followed.

a. Maintain the relations of modules from top to bottom.

b. Arrange the major activities of the problem in a hierarchical manner in such a way that the activities are represented by the nodes below the root node and connection between the root and the nodes is depicted by drawing lines.

c. Conceptualize each activity separately to determine how it can be divided into smaller subtasks.

A sample structure chart is presented in figure 7.6 in which top-level module "process_sales" communicates with three sub modules namely "validate_order", "do_classify_order" and "put_invoice" respectively.
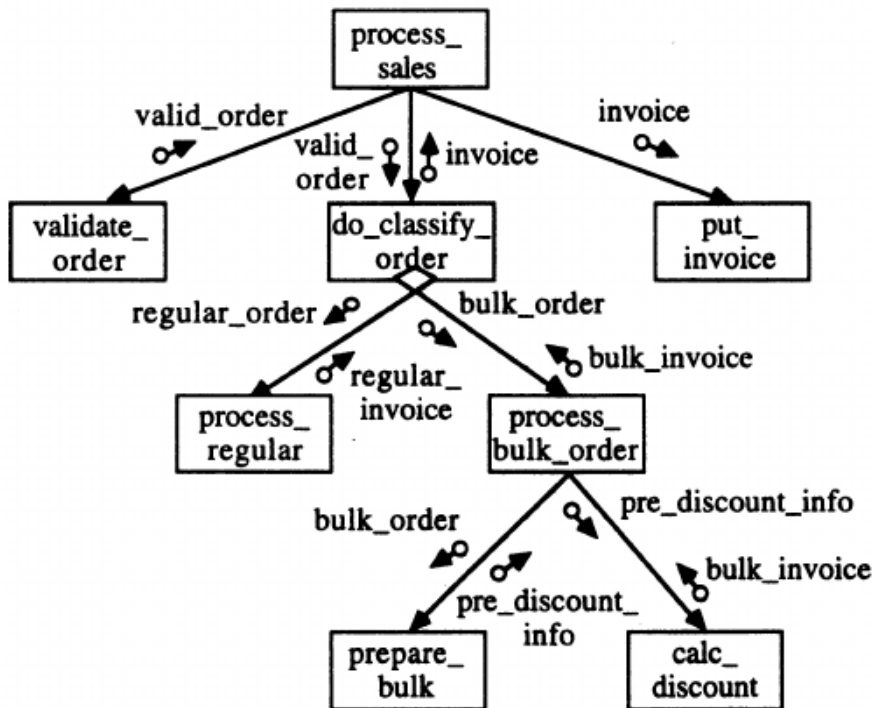

Figure 7.6 A sample structure chart

7.4.5 HIPO Diagram

HIPO (Hierarchical Input Process Output) diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970. A HIPO diagram generally consists of the following elements.

- A collection of top-level diagrams.

- A collection of detailed diagrams.

- A visual table of contents (VTOC) which consists of a tree or graph structured directory, summary of contents in each overview diagram, and a legend of symbol definitions.

Similar to an organization chart, HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure. Hence, HIPO diagrams can be used as a modelling tool in some environments.

Using the HIPO technique, designers can evaluate and refine a program's design, and correct flaws prior to implementation. Given the graphic nature of HIPO, users and managers can easily follow a program's structure.

A completed HIPO package has two parts:

A hierarchy chart is used to represent the top-down structure of the program. The hierarchy chart serves as a useful planning and visualization document for managing the program development process. A sample hierarchy chart is shown in the following figure 7.6 where a rectangular box represents a function which can further call its subfunctions.
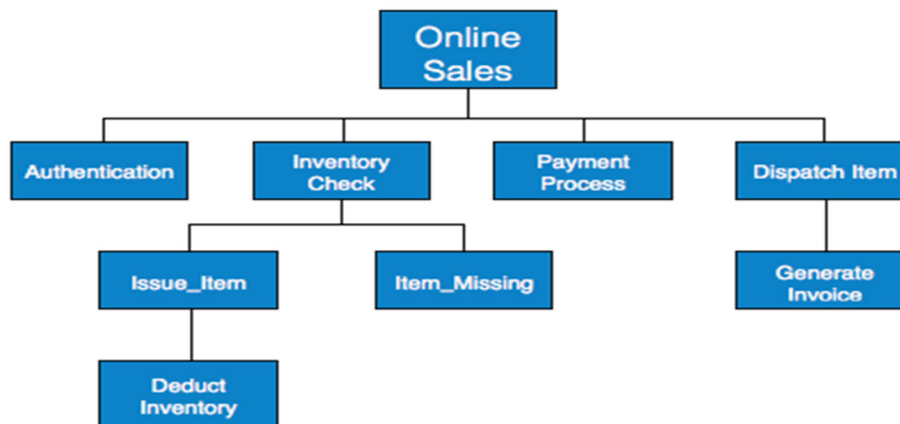

Figure 7.6 A Sample Structure chart

For each module depicted on the hierarchy chart, an IPO (Input-Process-Output) chart is used to describe the inputs to, the outputs from, and the process performed by the module. The IPO charts define for the programmer each module's inputs, outputs, and algorithms. For

example, IPO chart for Authentication function is presented in the figure 7.7.
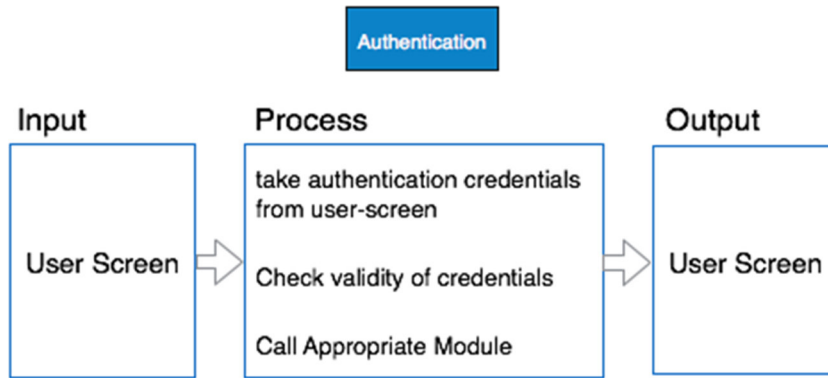

Figure 7.7 IPO chart for authentication function

7.4.6 Decision Tables

A decision table is a tabular representation of the logic of a problem. It specifies the possible conditions and the resulting actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

A decision table consists of three parts: Condition stubs which lists condition relevant to decision, Action stubs specify actions that result from a given set of conditions and Rules specify which actions are to be followed for a given set of conditions.

Creating Decision Table:

To create a decision table, the designer must follow basic four steps:

- Identify all possible conditions to be addressed,
- Determine actions for all identified conditions,
- Create Maximum possible rules,
- Define action for each rule.

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

Constructing a Decision Table:

PART 1. FRAME THE PROBLEM.

- Identify the conditions (decision criteria). These are the factors that will influence the decision.

- Identify the range of values for each condition or criteria.

- Identify all possible actions that can occur.

PART 2. CREATE THE TABLE.

- Create a table with 4 quadrants. (Put the conditions in upper left quadrant and place the actions in lower left quadrant)

- List all possible rules. Put the rules in the upper right quadrant.

- Enter actions for each rule. (In the lower right quadrant, determine what, if any, appropriate actions should be taken for each rule)

Example:

Let us take an example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while connecting to internet and their respective possible challenges. We include all possible problems under column conditions and the prospective actions under column Actions. Table 2 presents the decision table explaining in house internet connectivity problem.

Table 2: Decision Table – In-house Internet Troubleshooting

	Conditions/Actions	Rules							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

A. Choose the correct options from the followings:

1. Which of the following is not a component in DFD?
   A. Entities
   B. Attributes
   C. Process
   D. Data Flow

2. A data flow can
   A) Only emanate from an external entity
   B) Only terminate in an external entity
   C) May emanate and terminate in an external entity
   D) May either emanate or terminate in an external entity but not both

3. HIPO stand for
   A) Hierarchy input process output
   B) Hierarchy input plus output
   C) Hierarchy plus input process output
   D) Hierarchy input output Process

4. In a DFD external entities are represented by a
   A) Rectangle
   B) Ellipse
   C) Diamond shaped box
   D) Circle

5. After the design phase the document prepared is known as………………
   A) system specification
   B) performance specification
   C) design specification
   D) None of these

6. What is level 2 in DFD means?
   A) Highest abstraction level DFD is known as Level 2.
   B) Level 2 DFD depicts basic modules in the system and flow of data among various modules.
   C) Level 2 DFD shows how data flows inside the modules mentioned in Level 1.
   D) All of the above

7. The context diagram is also known as _____ .
   A) Level-0 DFD
   B) Level-1 DFD
   C) Level-2 DFD
   D) All of the above

8. .... is a tabular method for describing the logic of the decisions to be taken.
   A) Decision tables
   B) Decision tree
   C) Decision Method
   D) Decision Data

9. .... gives defining the flow of the data through an organization or a company or series of tasks that may not represent computerized processing.
   A) System flowchart
   B) Decision Tables
   C) System Trees
   D) Organization chart

10. The structure chart is
   A) a document of what has to be accomplished
   B) a statement of information processing requirement
   C) a hierarchical partitioning of the program
   D) All of the above

7.5 SUMMING UP

- In function-oriented design, the system is considered a group of many smaller sub-systems known as functions and hence, system is known as top view of all functions. These functions are capable of performing significant task in the system.
- Object Oriented design gives a detailed description of how the system can be built using objects.
- Design notations are used by the designers to represent software design. Various notations that are commonly used include flow charts, DFDs, HIPO diagram, Decision table, Structure chart, etc.
- A flow chart is a graphical design tool that depicts the sequence of operations to be carried out to solve a problem.

- As a graphical notation, DFD presents data sources, data sinks, data storage and processes performed on data as nodes, and logical flow of data as links between the nodes.
- A decision table is defined as a table that contains all the possible conditions for a problem and the corresponding results based on condition rules that connect condition with results. It is composed of rows and columns in the form of a matrix and matrix is formed in four quadrants.
- Structure Chart represent hierarchical structure of modules. It breaks down the entire system into lowest functional modules, describe functions and sub-functions of each module of a system to a greater detail. Structure Chart partitions the system into black boxes (functionality of the system is known to the users but inner details are unknown).
- A HIPO diagram comprises of a hierarchy chart that pictorially represents control structure of a program and a set of IPO (Input-Process-Output) charts that describe the inputs to, the outputs from, and the functions (or processes) performed by each module on the hierarchy chart.

7.6 ANSWERS TO CHECK YOUR PROGRESS

1. B	2. C	3. A	4. A	5. C
6. C	7. A	8. A	9. A	10.C

7.7 POSSIBLE QUESTIONS

1. Compare relative advantages of the object-oriented and function-oriented approaches to software design.

2. How do you describe the term top-down decomposition in the context of function-oriented design?

3. Mention the differences between a structure chart and a flow chart.

4. Explain how the DFD model can help to understand the working of a software system.

5. Draw the context diagram and level 1 DFD for Students' academic record management system.

6. Write the main advantages of using decision table in designing a problem.

UNIT 8: SOFTWARE DESIGN III

Unit Structure:

8.1 INTRODUCTION

Software design is a method to abstract the software requirements into software implementation. It takes the user requirements as challenges and tries to find an optimal solution. There are several alternatives of software design methodology. In this chapter, we will discuss two of them i.e., Structured Design Methodology and Object-Oriented Design methodology

8.2 UNIT OBJECTIVES

At the end of this lesson the student will be able to

- *Identify* the aim of structured design.
- *Explain* what a structure chart is.
- *Differentiate* between a structure chart and a flow chart.
- *Identify* the activities carried out during transform analysis with examples.
- *Explain* what is meant by transaction analysis.
- *Identify* the basic difference between object-oriented analysis (OOA) and object-oriented design (OOD).
- *Explain* what a model and how models are useful.
- *Explain* what UML means.
- *Identify* different types of views captured by UML diagrams.
- *Explain* the utility of different types UML diagrams.

8.3 STRUCTURED DESIGN METHODOLOGY

Structured design transforms the results of the structured analysis into a structure chart. Structured analysis is a well-organized development process that uses graphical tools to analyse and improve the objectives of an existing system and to develop a new system specification. It allows understanding the system and its activities in a logical way. Data Flow Diagrams (DFD), Data Dictionary, Decision Trees, Decision Tables, Pseudocode, etc. are some examples of tools available for structured analysis.

On the other hand, a structure chart represents the architecture of the software i.e., the dependency and the parameters passed among the various modules of the software system. There are two strategies to converts the results of the structured analysis into a structure chart.

- Transform analysis and

- Transaction analysis

At each level of conversion, it is very important to determine whether the transform analysis or the transaction analysis is suitable for a particular structured analysis or not.

8.3.1 Building Blocks for Structure Chart

The basic building blocks that are used to design structure charts are the following:

- **Rectangular boxes:** Used to represents the process or task of the system. It is of three types - Control Module, Sub Module and Library Module. A control module branches to more than one sub-module. Library modules are reusable and invokable from any module.
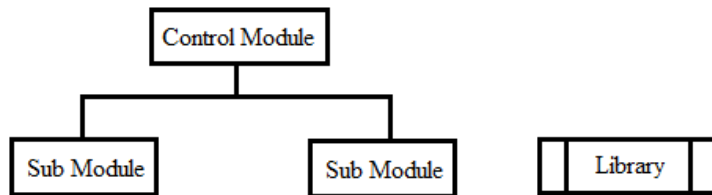
Fig 8.1 Representation of Modules

- **Module invocation arrows:** These are module connecting arrows. The direction of the arrow indicates that control is passed from one module to another module.
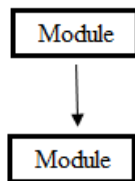
Fig 8.2 Module invocation arrows

- **Data flow arrows**: It represents data passes from one module to another module in the direction of the arrow. It is represented by a directed arrow with an empty circle at the end.
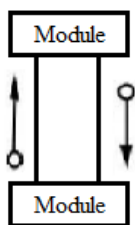
Fig 8.3 Data flow arrows

- **Control flow arrows:** It represents the flow of control between the modules. It is represented by a directed arrow with a filled circle at the end.
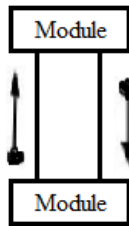
Fig 8.4 Control flow arrows

- **Selection or Condition:** Denoted by a diamond symbol. It represents that the control module can select any of the submodules on the basis of some condition.
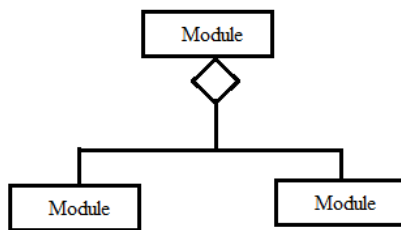

Fig 8.5 Selection

- **Repetition or Loop:** It signifies the repetitive execution of a module by the submodule. A curved arrow is used to represents a loop in the module.
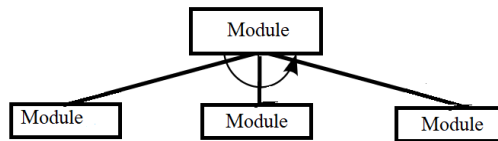

Fig 8.6 Loop

- **Physical Storage:** Physical Storage is that where all the information is to be stored.
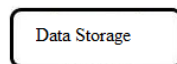

Fig 8.7 Physical Storage

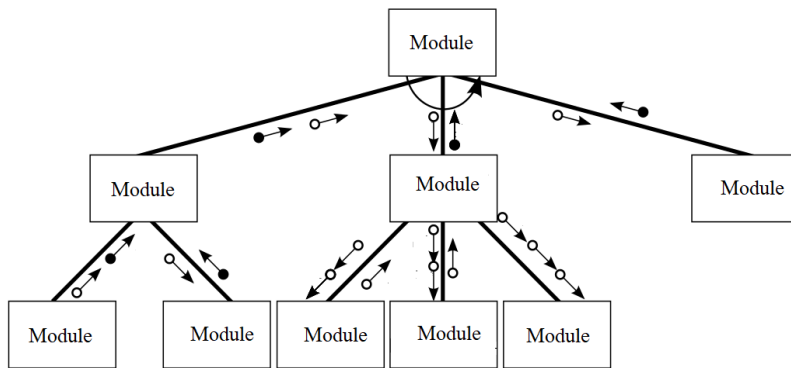The following figure shows a layout of structure chart

Fig 8.8 Physical Storage

We are used to flow chart representation of a program. Though looks similar, a structure chart differs from a flow chart. It is typically difficult to recognize the different modules of the software from its flow chart representation. Besides, data interchange among different modules is not represented in a flow chart.

8.3.2 Transform Analysis

Transform analysis distinguishes the key functional modules and the high-level inputs and outputs for these modules. Steps of Transform Analysis are given below

Step 1: The first step in transform analysis is to divide the structured analysis into 3 types of parts:

- Input

- Logical processing

- Output

The input portion comprises procedures that convert input data from physical form (e.g. character from terminal) to logical forms (e.g. internal data lists, tables etc.). Each input portion is known as an afferent branch.

The output portion of transform analysis alters output data from logical to physical form. Each output portion is termed an efferent branch.

The remaining portion of transform analysis is called the central transform.

Step 2: In the next step, the structure chart is derived by drawing one functional component for the central transform, and the afferent and efferent branches.

Each input and output unit are represented as boxes in the first level structure chart. Processes that perform logical processing like sorting or filtering input data are part of central transforms. Each central transform is depicted as a single box. Processes like input validation or adding information to input are not central transforms.

Step 3: In this step, sub-components or sub-functions are added to each of the high-level functional components (if required). It makes a structured chart more refined. This process of breaking high-level functional components into sub-components is called factoring. Many levels of subcomponents may be added. The factoring process is continued until all components of structured analysis are represented in the structure chart.

Example: Structure Chart of a simple email server

We can see that three basic functions that the email server needs to perform – accept the login details from the user, validate users, then login to the mailbox. The level 1 DFD of this system may be as follows
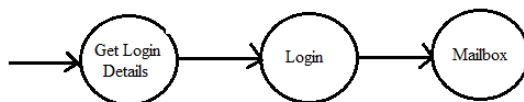

Fig 8.9 Level 1 DFD of email server

From this DFD, we can get the structure chart shown in the following figure by applying all 3 steps of transform analysis mentioned above.
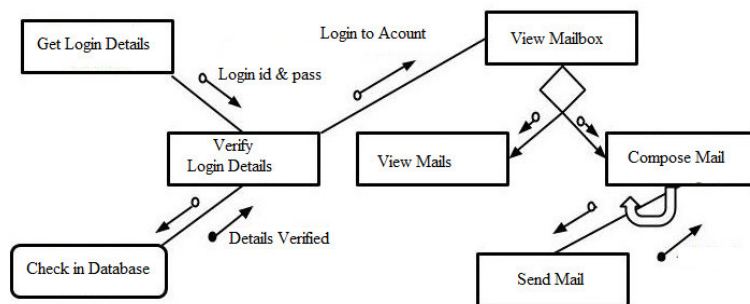

Fig 8.10 Structure chart of email server

8.3.3 Transaction Analysis

Transaction analysis is an alternative structured design strategy for developing structure charts. It is useful while designing transaction processing programs. One of several possible paths through the data flow diagram is pass through depending upon the input data item.

Each different way in which input data is handled is considered a transaction. The input data is traced to the output for each identified transaction. All the traversed bubbles of data flow diagram belong to the transaction and these bubbles should be mapped to the same module on the structure chart. Initially we draw a root module in the structured chart and each identified transaction will be drawn below this root module. Every transaction is associated with a tag, which identifies its type. This tag is used to divide the system into transaction modules and a transaction center module during transaction analysis.

It is also possible that some transactions may not require any input data. Such kind of transactions can be identified by practicing a large number of examples.

STOP TO CONSIDER

A DFD model of a system graphically portrays the conversion of the data input to the system to the final result through a hierarchy of levels. It starts with the most abstract definition of the system (context diagram) and at each higher level DFD, more details are successively introduced. To develop a higher-level DFD model, processes are decomposed into their sub-processes and the data flow among these sub-processes is identified.

CHECK YOUR PROGRESS

1. Differentiate between a structure chart and a flow chart

2. Which documents are produced at the end of structured analysis activity?

3. For the following, mark all options which are true

    a. The purpose of structured analysis is

        i.   to capture the detailed structure of the system as perceived by the user

        ii.   to define the structure of the solution that is suitable for implementation in some programming language

iii.   all of the above

b. Structured analysis technique is based on

i.   top-down decomposition approach

ii.   bottom-up approach

iii.   divide and conquer principle

iv.   none of the above

c. In a structure chart, a module represented by a rectangle with double edges is called

i.   root module

ii.   library module

iii.   primary module

iv.   none of the above

d. Which of the following types of bubbles in DFD may belong to the central transform ?

i.   input validation

ii.   adding information to the input

iii.   sorting input

iv.   filtering data

e. The input portion in the DFD that transform input data from physical to logical form is called

i.   central transform

ii.   efferent branch

iii.   afferent branch

iv.   none of the above

8.4 OBJECT-ORIENTED MODELLING

8.4.1 Concept of Object-Oriented Modelling

Object-Oriented (OO) Modelling is a way of thinking about problems using models organized around real-world concepts. In this approach, the system is viewed as a collection of objects. Object have their own data which defines their states and functions or operations to work on

these data. For example, in a Banking Software, each account may be a separate object with its own data like account holder's name, balance amount, and functions like suspend(), deposit(), withdraw(), update() to operate on these data.

Each object is said to be an instance of some class. A class can be assumed as generalized description of an object. Class can be termed as the blueprint from which individual objects are created. Classes may inherit data and functions from other classes. But functions defined for one object cannot refer to or change the data of other objects. However, the same function may behave differently in different classes.

8.4.2 Object-Oriented Analysis vs Object-Oriented Design

The first technical activity performed as part of object-oriented modelling is Object-Oriented Analysis (OOA). It is used to develop an initial analysis model of the system from the requirements specification. This analysis model is then transformed by Object-Oriented Design (OOD) into a design model that works as a plan for software development. Object-Oriented Design (OOD) techniques not only identify objects but also identify the internal details of the objects and the relationships existing among these objects.

8.5 OBJECT MODELLING USING UML

Designers build different kinds of models for various purposes before constructing things. The main reason for constructing models is to deal with systems that are too complex to understand directly. Models helps to reduce complexity by separating out a small number of important things to deal with at a time. Once models of a system have been developed, it can be used for a variety of purposes during software development, including the following:

- Analysing and Specification
- Reduction of Complexity
- Designing and testing the system
- Better understanding of the problem

Since a model can be used for a variety of purposes the model may vary depending on the purpose for which it is being created. For example, a model constructed for initial analysis and specification will be different from the model constructed for design.

UML (Unified Modelling Language) is a standard language for specifying, visualizing, constructing, and documenting software systems. It was created by the Object Management Group (OMG). UML 1.0 specification draft was proposed to the Object Management Group (OMG) in 1997. Initially it was started to capture the behavior of complex software and non-software system and now it has become an OMG standard. UML has its own syntax and semantics to create a visual model of the system. UML is not a programming language. It is basically used to document object-oriented and analysis results obtained using some methodology. There are many tools which can be used to produce code in different programming languages using UML diagrams.

The conceptual model of UML can be mastered by learning the following three major elements

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

The building blocks of UML can be defined as – *Things*, *Relationships* and *Diagrams*

8.5.1 Things in UML

Things are the most vital building blocks of UML. It can be – *Structural, Behavioral, Grouping* and *Annotation* .

A) Structural Things: Structural things represent the physical and conceptual elements of a system. Brief descriptions of various structural things are given below.

- **Class:** Class denotes a set of objects having similar characteristics.
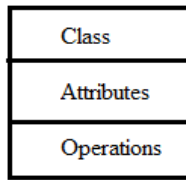
Fig 8.11 : Representation of Class

- **Interface:** Interface defines a set of operations, which specify the properties of a class.
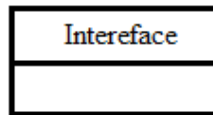
Fig 8.12 : Representation of Interface

- **Collaboration:** Collaboration describes interactions between elements.

Fig 8.13 : Representation of Collaboration

- **Component:** Component defines the physical part of a system.

Fig 8.14 : Representation of Component

- **Node:** Physical elements that exists at run time are termed as node

Fig 8.15 : Representation of Node

B) Behavioral things : A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things −

- **Interaction :** Interaction is defined as a behavior that involves a group of messages exchanged among elements to complete a specific job.

Message
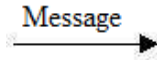
Fig 8.16: Representation of Interaction
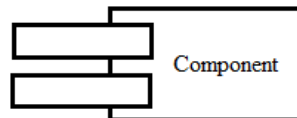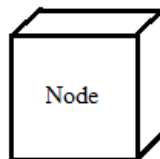
- **State machine :** It defines the sequence of states an object goes through in response to external factors responsible for a state change.
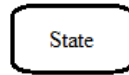
State

Fig 8.17 : Representation of State

C) Grouping Things : Grouping things can be defined as a technique to group elements of a UML model together. There is only one grouping thing available i.e. **package** which gathers structural and behavioral things.
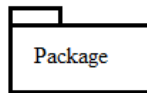
Package

Fig 8.18 : Representation of Package

D) Annotational Things : Annotational things is a mechanism to capture remarks, descriptions, and comments of UML model elements. Note is the only one Annotational thing available in UML Modelling.
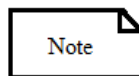
Note

Fig 8.19 : Representation of Note

8.5.2 Relationship

Relationship demonstrates how the elements of a system are associated with each other. This association between elements describes the functionality of a system. There are four types of relationships – *Dependency*, *Association*, *Generalization* and *Realization.*

- **Dependency** defines a relationship between two things in which any alteration in one element also affects the other element.

Dependancy
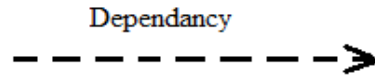
Fig 8.20 : Representation of Dependency

- **Association** describes how many objects are taking part in a relationship. It is basically a set of links that attaches the elements of a UML model.
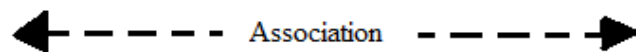
Association

Fig 8.21 : Representation of Association

- **Generalization** can be defined as a relationship that links a specialized element with a generalized element. It generally describes the inheritance relationship between objects.
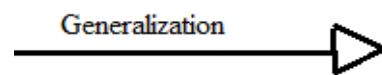
Generalization

Fig 8.22 : Representation of Generalization

- **Realization** can be viewed as a relationship in which two elements are connected. One element defines some characteristics which are not implemented and the other element implements them. This kind of relationship exists in the case of interfaces.
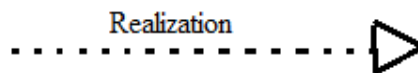
Realization

Fig 8.23 : Representation of Realization

8.5.3 UML Diagrams

All the elements, relationships can be associated in different ways to make a complete UML picture, which is known as diagram. UML diagrams helps us to understand the system in a better and simple way. Usually diagrams are made in an incremental and iterative way. A single diagram is not adequate to cover all the characteristics of the system. There are two broad categories of diagrams i.e. *Structural Diagrams* and *Behavioral Diagrams*. They are again divided into subcategories.
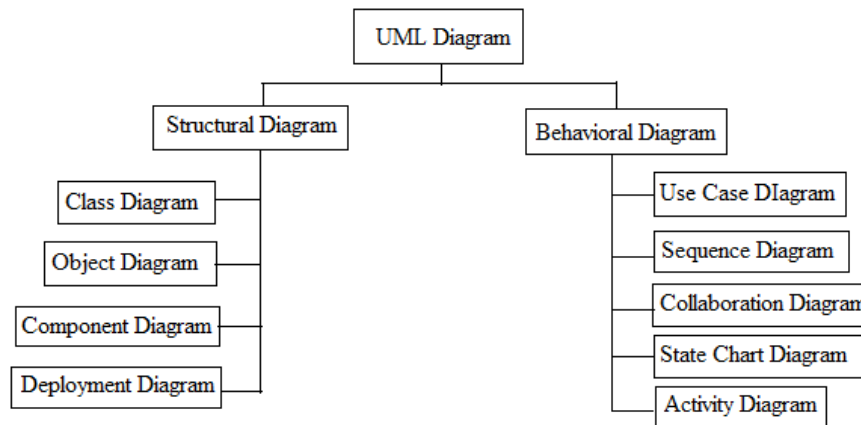
Fig 8.24: Categories of UML Diagram

Different UML diagrams provide different perspectives of the software system to be designed and developed. They facilitate a comprehensive understanding of the system. The UML diagrams can capture the following five views of a system:

- **User's view**: This view defines the functionalities (facilities) made available by the system to its users. The users' view captures the external users' view of the system in terms of the functionalities offered by the system.

- **Structural view**: The structural view describes the classes of objects important to the understanding of the working and implementation of the system. It also defines the relationships among the objects.

- **Behavioral view**: The behavioral view captures how objects interact with each other to realize time dependant or dynamic behavior of the system.

- **Implementation view**: Implementation view captures the important components of the system and their dependencies.

- **Environmental view**: Environmental view models how the different components are

  implemented on different pieces of hardware.

8.5.3.1 Structural Diagrams

The structural diagrams depict the main structure of a system represented by classes, interfaces, objects, components, and nodes. The four structural diagrams are −

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

A) Class Diagram

Class diagrams are the most common diagrams used in UML. A class diagram consists of classes, interfaces, associations, and collaboration. Class diagrams basically represent the object-oriented view of a system

Class diagrams have a lot of properties to consider while drawing. The following points should be remembered while drawing a class diagram −

- Name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and its relationships should be identified properly in advance.
- Attributes and methods of each class should be clearly identified
- Unnecessary properties should be avoided, otherwise they will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram so that it becomes easily understandable to the developer/coder.
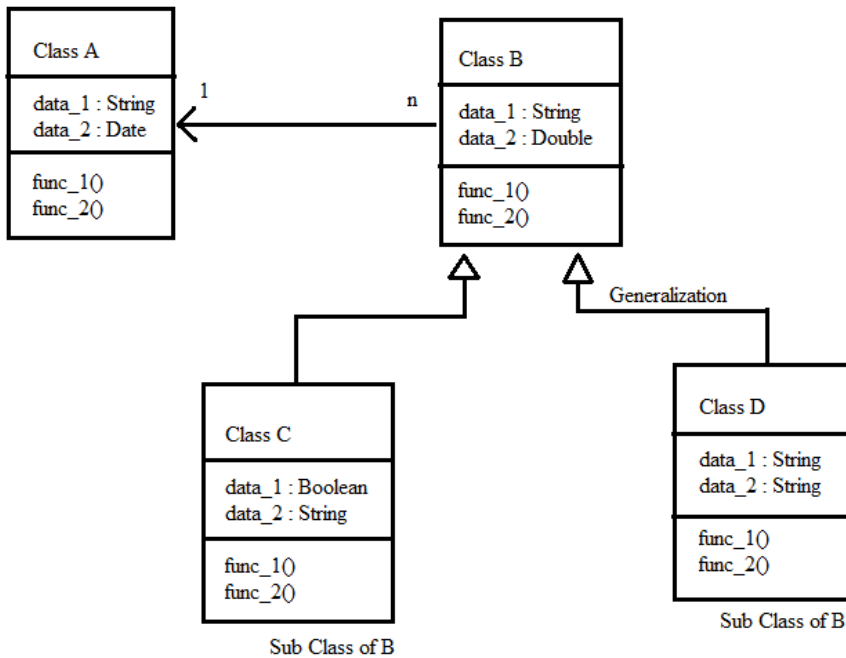
Fig 8.25 : Class Diagram

UML diagrams are not directly mapped with any object-oriented programming languages but the class diagram is an exception. As it clearly shows the mapping with object-oriented languages.

B) Object Diagrams

Object diagrams are derived from class diagrams. Object diagrams help to render a set of objects and their relationships as an instance. Objects and links are the two essential elements used to construct an object diagram. The object diagram should have a meaningful name to indicate its purpose and association among various objects should be clarified.
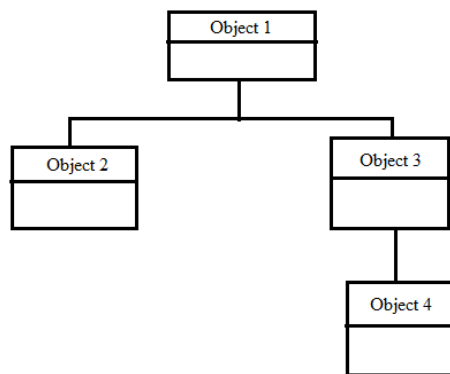
Fig 8.26 : Object Diagram

C) Component Diagram

The purpose of the component diagram is different from a class diagram or object diagram. It describes the components used to make functionalities of the system rather than describing those functionalities. Component diagrams are used to model the physical aspects such as executables, libraries, files, documents, etc. of a system
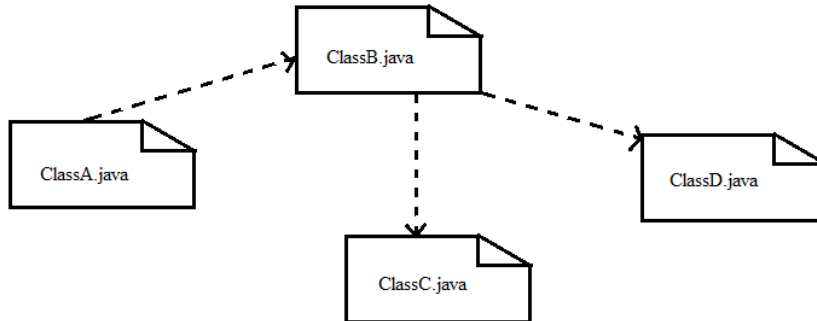
Fig 8.27 : Component Diagram

D) Deployment Diagram

Deployment diagrams are a set of nodes and their relationships. These nodes are nothing but physical hardware used to deploy the application.
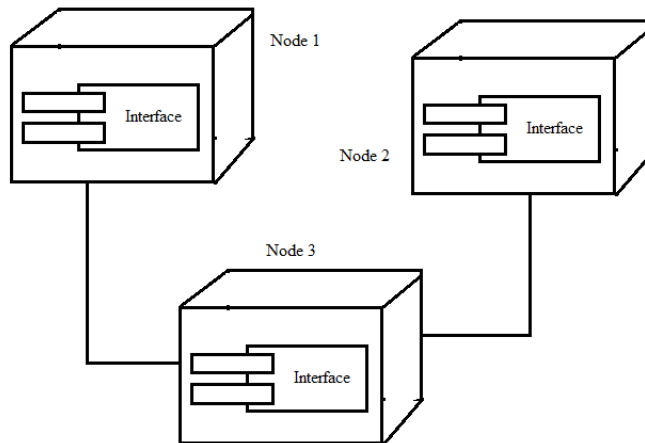
Fig 8.28 : Deployment Diagram

8.5.3.2 Behavioral Diagrams

Behavioral diagrams basically capture the dynamic aspect i.e. the behavior of the system when it is in operational or running state. UML has the following five types of behavioral diagrams

- Use case diagram

- Interaction diagram

- State chart diagram

- Activity diagram

A) Use case diagrams

A use case basically represents a sequence of interactions between the system and users. Use case diagrams are used to gather the requirements of a system including internal and external effects. These requirements are typically design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. Actors are nothing but an entity that interacts with the system. It can be a human user, some internal applications, or maybe some external applications. While planning to draw a use case diagram, the following items should be identified in advanced

- Functionalities to be represented as a use case

- Actors and

- Relationships among the use cases and actors

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, the following guidelines need to be followed to draw an efficient use case diagram

- The name should be chosen in such a way so that it can identify the functionalities performed.

- Give a appropriate name for actors.

- Relationships and dependencies should be clearly visible in the diagram.

- Use notes (if required) to clarify important points.
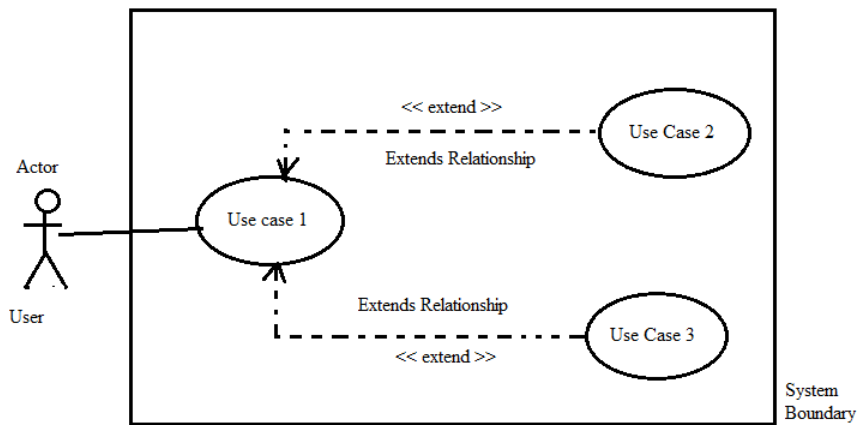
Fig 8.29: Use Case Diagram

B) Interaction Diagrams

Interaction diagrams visualize the interactive behavior of a system. Visualizing the interaction is not an easy task. Hence, different types of models are used to capture the various aspects of the interaction. This interaction is a part of the dynamic behavior of the system and it is represented in UML by two diagrams known as *Sequence diagram* and *Collaboration diagram*. The basic purpose of both the diagrams is similar.

A *sequence diagram* shows the message sequence of various objects The message flow is nothing but a method call of an object.
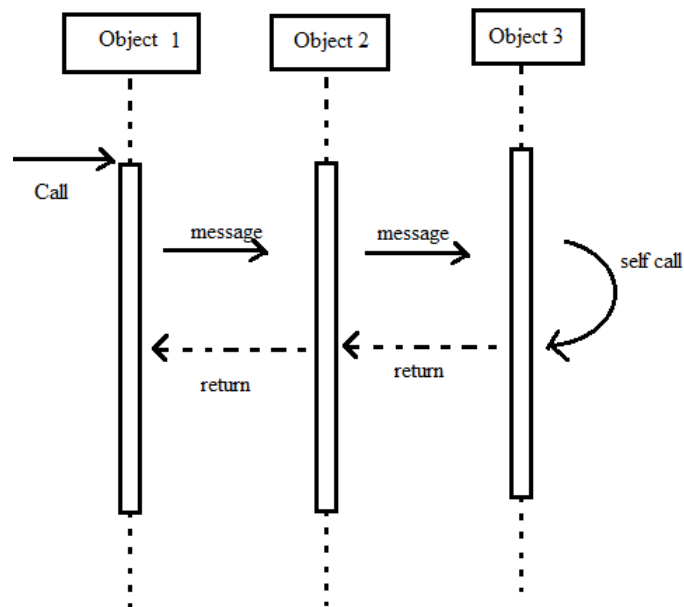
Fig 8.30 : Sequence Diagram

In the *collaboration diagram*, some numbering technique is used to indicate the method call sequence. The number specifies how the methods are called one after another.
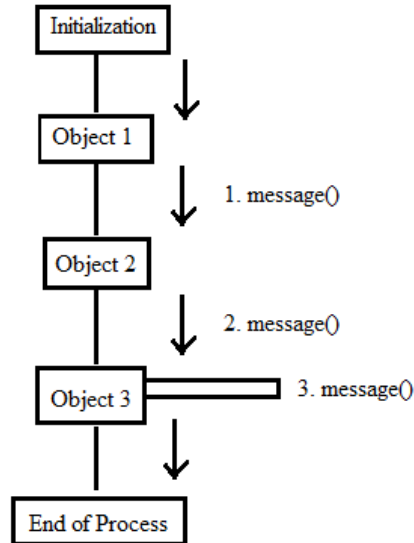
Fig 8.31 : Collaboration Diagram

C) Statechart Diagram

Statechart diagram is another type of UML diagrams which is used to model the dynamic nature of a system. Statechart diagrams define different states of an object throughout its lifetime and these states are changed by events. Statechart diagrams are convenient to model systems that respond to external or internal events.
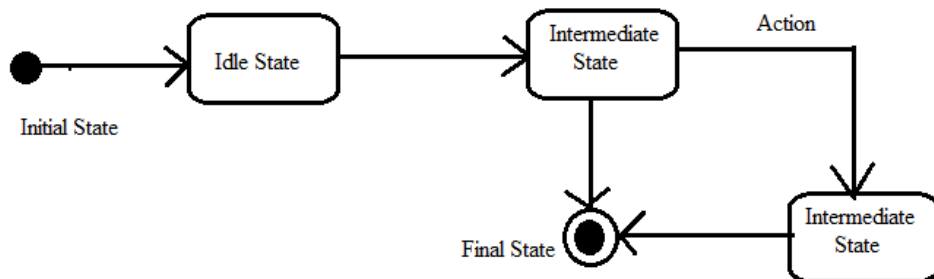
Fig 8.32 : Statechart Diagram

D) Activity Diagram

Activity diagram is fundamentally a flowchart to represent the flow from one activity to another. The activity can be defined as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

Activity diagram has more impact on understanding the system rather than on implementation details.

Fig 8.33 : Activity Diagram

    ii.   an object-oriented development methodology

    iii.   an automatic code generation tool

    iv.   none of the above

b. In the context of use case diagram, the stick person icon is used to represent

    i.   human users

    ii.   external systems

    iii.   internal systems

    iv.   none of the above

c. Which of the following view captured by UML diagrams can be considered as

black box model of a system?

    i.   structural view

    ii.   behavioral view

    iii.   user's view

    iv.   implementation view

8.6 SUMMING UP

Software design is a process to convert user requirements into some suitable form, which helps the programmer in software coding and implementation. Structured design methodology is typically based on 'divide and conquer' approach where a problem is fragmented into several small problems and each small problem is separately solved until the whole problem is solved. A good structured design has high cohesion and low coupling arrangements. Object oriented design works around the entities and their features instead of functions involved in the software system. UML is a modelling language used to model software and non-software systems. Although UML is used for non-software systems if we look into UML diagrams all would basically be designed based on the objects.

Hence, the relation between Object - Oriented design and UML is very important to understand. Before understanding the UML in detail, the Object - Oriented concept should be learned properly.

8.7 ANSWERS TO CHECK YOUR PROGRESS

1. A structure chart differs from a flow chart in following ways:

- It is usually difficult to identify the different modules of the software from its flow chart representation.

- Data interchange among different modules is not represented in a flow chart.

2. Graphical representation of structured analysis results Data Flow Diagrams.

3. a. i

    b. i, ii

    c. ii

    d. iii, iv

    e. iii

4. An important reason behind constructing a model is that it helps manage complexity. Once models of a system have been constructed, these can be used for a variety of purposes during software development, including the following:

- Analysing and Specification

- Reduction of Complexity

- Designing and testing the system

- Better understanding of the problem

Since a model can be used for a variety of purposes, it is reasonable to expect that the model would vary depending on the purpose for which it is being constructed. For example, a model developed for initial analysis and specification should be very different from the one used for design. A model that is being used for analysis and specification would not show any of the design decisions that would be made later on during the design stage. On the other hand, a model used for design purposes should capture all the design decisions. Therefore, it is a good idea to explicitly mention the purpose for which a model has been developed, along with the model.

5. The behavioral view is captured by the following UML diagrams:

- Sequence diagrams

- Collaboration diagrams
- State chart diagrams
- Activity diagrams

6. Structural aspects of a system are captured by the following UML diagrams:

- Class diagrams
- Object diagrams

7. Implementation view captures the important components of the system and their dependencies.

8.

a. False.   A state chart diagram is normally used to model how the state of an object changes in its life time. State chart diagrams are good at describing how the behavior of an object changes across several use case executions.

b. False.   Interaction diagrams are models that describe how groups of objects team up to realize some behavior. Typically, each interaction diagram comprehends the behavior of a single use case. An interaction diagram shows a number of example objects and the messages that are passed between the objects within the use case.

c. False. A class diagram describes the static structure of a system. It shows how a system is structured rather than how it behaves. The static structure of a system comprises of a number of class diagrams and their dependencies.

9.   a. iv

b. i

c. iii

8.8 POSSIBLE QUESTIONS

1. Identify different types of views of a system captured by UML diagrams.

2. What is the basic difference between object-oriented analysis (OOA) and object-oriented design (OOD) ?

3. What is the need for developing use case diagram?

4. Differentiate Activity diagram and State chart diagram.

5. What do you understand by relationships in UML?

6. What are the advantages of creating a Model ?

8.9 REFERENCES AND SUGGESTED READINGS

- *Fundamentals of Software Engineering, Fifth Edition, 2018,* Rajib Mall; PHI

- *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition

BLOCK II:
SOFTWARE CODING, TESTING AND MAINTENANCE

UNIT 1: SOFTWARE CODING

Unit Structure:

1.0   INTRODUCTION

In this unit, you will learn the coding phase of software development life cycle. This unit covers the basic understanding of the coding activities involved during software development. The goal of the coding activity is to implement the design in the best possible manner. It affects both the testing and the maintenance process profoundly. As we know, the time spent in the coding activity is the small percentage of the total software development cost and the testing and the maintenance consumes the major percentage. Thus, it is very much clear that the purpose of coding phase is not to reduce the implementation cost but to reduce the cost of the later phase. In other words, the goal during this phase is not to simplify the programmers job rather the goal should be to simplify the job of tester and the maintainer. It is very important to understand that while coding it must be kept in mind that the program should not be

constructed so that they are easy to write, but so that they are easy to write and understand.

1.2 UNIT OBJECTIVES

After going through the unit you will be able:

- To understand the coding phase of software development life cycle.

- The importance of coding standards and guidelines followed by organizations.

- The difference between coding standards and guidelines.

- The different coding methodologies followed in the software industries.

- The concept of coding verification and the also discusses about the importance of code documentation.

1.3 CODING STANDARDS AND GUIDELINES

Writing an efficient software code requires a thorough knowledge of programming. This knowledge can be implemented by following a coding style which comprises several guidelines that help in writing the software code efficiently and with minimum errors. These guidelines, known as **coding guidelines,** are used to implement individual programming language constructs, comments, formatting, and so on. These guidelines, if followed, help in preventing errors, controlling the complexity of the program, and increasing the readability and understandability of the program. A set of comprehensive coding guidelines encompasses all aspects of code development. To ensure that all developers work in a harmonized manner (the source code should reflect a harmonized style as a single developer had written the entire code in one session), the developers should be aware of the coding guidelines before starting a software project.

Moreover, coding guidelines should state how to deal with the existing code when the software incorporates it or when maintenance is performed. Since there are numerous programming languages for writing software codes, each having different features and capabilities, coding style guidelines differ from one language to another. However, there are some basic guidelines, which are followed in all programming languages. These include naming conventions, commenting conventions, and formatting conventions.

Good software development organizations usually develop their own coding standards and guidelines depending on what suits there organization best and based on the specific type of product they develop. In the following section, we shall only list some general coding standards and guidelines, which are commonly adopted by much software development organization, rather than trying to provide an exhaustive list.

A coding standard lists several rules to be followed during coding such as the way variables are to be named, the way the code is to be laid out, the error return conventions etc. Besides the coding standards, several coding guideline are to be prescribed by software companies. But what is the difference between coding standard and coding guidelines?

Coding standards have to be mandatorily followed by the programmers, and compliance to coding standards is verified before the testing phase can start. In contrast , coding guidelines provide some general suggestions regarding the coding style to be followed but leave the actual implementation of these guidelines to the discretion of the individual developers.

1.3.1 Representative Coding Standards

a) Rules for limiting the scope of global variables: These rules list what types of data can be declared global and what cannot, with a view to limit the data that needs to be defined with the global scope.

b) Standard headers to precede the code of different modules: The information contained in the headers of the of different modules should be standard for an organization and exact format of the header must be

specified. The following is an example of some of the header format adopted by the companies:

- Name of the module

- Date on which the module was created

- Authors name

- Modification history

- Synopsis of the module

- Different functions supported in the module, along with their input/output parameters

- Global variables accessed / modified by the module

c) Naming conventions for global variables, local variables and constant identifiers: The variables are named using mixed case lettering. Global variable names should always start with a capital letter (e.g GlobalData) and local variable name would start with small letters (e.g localData). Constant names should be formed using capital letters only (e.g CONSTDATA).

d) Conventions regarding error return values and exception handling mechanism: The way error conditions are reported by different functions in a program should be standard within an organization. For example all functions while encountering an error condition should either return 0 or 1 consistently, independent of which the program has written a code.

1.3.2 Representative Coding Guidelines

The following are some representatives coding guidelines that are recommended by many software development organizations:

a) Do not use a coding style that is too clever or too difficult to understand: Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code. Clever coding can obscure meaning of the code and hamper understanding. As a result, it can make maintenance and debugging difficult and expensive.

b) **Avoid obscure side effects:** The side effect of a function call include modifications to the parameters passed by reference, modifications of global variables and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code. Obscure side effects make it difficult to understand a piece of code. For example , if a global variable is changed or some file I/O performed obscurely in a called module, it becomes difficult to infer from the functions name and the header information, making it difficult to understand the code.

c) **Do not use an identifier for multiple purposes:** Programmers often use the same identifier to denote several temporary entities. For example, some programmers use a temporary loop variable for also computing and storing the final result. The reason for doing so is that if same variable is used for multiple purpose it can save memory since they are using the same memory location and if we used three different variables then it will take three different memory locations. However, there is several things wrong with this approach and hence should be avoided. Some of the problems caused by use of variable for multiple purpose are as follows :

- Each variable should be given a descriptive name indicating its purpose. Use of single variable for multipole purpose may lead to confusion and reduces the understandability of the code.

- Use of single variable for multiple purpose usually makes future enhancement more difficult. For example, while changing the final computed result from integer to float type, the programmer might subsequently notice that it has been used as a temporary loop variable that cannot be float type.

d) **The code should be well documented:** As a rule of thumb, there should be at least one comment line on the average for every three-source lines of code.

e) **The length of any function should not exceed 10 source lines:** A lengthy function is usually very difficult to understand as it probably has a large number of variables and carries out many different types of computations. It may carry large number of bugs.

f) **Do not use GO TO statements:** Use of GO TO statement makes the program unstructured, thereby making it difficult to understand, debug and maintain the program.

CHECK YOUR PROGRESS

1.   Coding instruction in computer language is known as …………………

2.   Coding guidelines helps in ….. and helps to detect ….. easily.

3.   Code should be well documented.

4.   The coding standards decreases the efficiency of the programmers.

1.4   CODING METHODOLOGIES

1.4.1 Code Review

It in undertaken after the modules is successfully compile, it means all the syntax error have been eliminated from the module. The code reviews are cost effective strategies for eliminating coding errors and for producing high quality code. The reason behind why code review is much more cost effective strategy to eliminate errors from code compare to testing since it directly detect errors. On the other hand testing only detect failures and significant effort need to be given in debugging to locate the error.

Normally there are two types of reviews carried out on the code of a module.

a)   *Code Walkthrough*

Code Walkthrough is a form of peer review in which a programmer leads the review process and the other team members ask questions and spot possible errors against development standards and other issues.

- The meeting is usually led by the author of the document under review and attended by other members of the team.

- Review sessions may be formal or informal.

- Before the walkthrough meeting, the preparation by reviewers and then a review report with a list of findings.

- The scribe, who is not the author, marks the minutes of meeting and note down all the defects/issues so that it can be tracked to closure.

- The main purpose of walkthrough is to enable learning about the content of the document under review to help team members gain an understanding of the content of the document and also to find defects.

b) *Code Inspection*

Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage.

- The main purpose of code inspection is to find defects and it can also spot any process improvement if any.

- An inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.

- Preparation before the meeting is essential, which includes reading of any source documents to ensure consistency.

- A trained moderator, who is not the author of the code, often leads inspections.

- The inspection process is the most formal type of review based on rules and checklists and makes use of entry and exit criteria.

- It usually involves peer examination of the code and each one has a defined set of roles.

- After the meeting, a formal follow-up process is used to ensure that corrective action is completed in a timely manner.

Common programming errors that can be checked during code Inspection.

The following is the list of classical programming errors that needs to be considered during code inspection:

- Use of uninitialized variables

- Jumps into loops

- Non – terminating loops

- Incompatible assignments

- Array indices out of bounds

- Improper storage allocation and deallocation.

- Mismatches between actual and formal parameter in procedure calls

- Use of incorrect logical operators or incorrect precedence among operators

- Improper modification of loop variables

- Comparison of equality of floating point values , etc

1.4.2 Code Verification

Code verification is the process used for checking the software code for errors introduced in the coding phase. The objective of code verification process is to check the software code in all aspects. This process includes checking the consistency of user requirements with the design phase. Note that code verification process does not concentrate on proving the correctness of programs. Instead, it verifies whether the software code has been translated according to the requirements of the user.

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.

- Verification answers the question– "Are we developing this product by firmly following all design specifications?"

- Verifications concentrates on the design and system specifications.

Target of the test are -

- **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.

- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.

- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

1.4.2.1 Static and Dynamic Techniques

The code verification techniques are classified into two categories, namely, dynamic and static. The **dynamic technique** is performed by executing some test data. The outputs of the program are tested to find errors in the software code. This technique follows the conventional approach for testing the software code. In the **static technique,** the program is executed conceptually and without any data. In other words, the static technique does not use any traditional approach as used in the dynamic technique. Some of the commonly used static techniques are code reading, static analysis, symbolic execution, and code inspection and reviews.

a)   **Code Reading:** Code reading is a technique that concentrates on how to read and understand a computer program. It is essential for a software developer to know code reading. The process of reading a software program in order to understand it is known as **code reading or program reading.** In this process, attempts are made to understand the documents, software specifications, or software designs. The purpose of reading programs is to determine the correctness and consistency of the code. In addition, code reading is performed to enhance the software code without entirely changing the program or with minimal disruption in the current functionality of' the program. Code reading also aims at inspecting the code and removing (fixing) errors from it.Code reading is a passive process and needs concentration. An effective code reading activity primarily focuses on reviewing 'what is important'. The general conventions that can be followed while reading the software code are listed below.

▪   *Figure out what is important*: While reading the code, emphasis should be on finding graphical techniques (bold, italics) or positions (beginning or end of the section). Important comments may be highlighted in the introduction or at the end of the software code. The

level of details should be according to the requirements of the software code.

▪ ***Read what is important***: Code reading should be done with the intent to check syntax and structure such as brackets, nested loops, and functions rather than the non-essentials such as name of the software developer who has written the software code.

b) **Static Analysis**: Static analysis comprises a set of methods used to analyze the source code or object code of the software to understand how the software functions and to set up criteria to check its correctness. Static analysis studies the source code without executing it and gives information about the structure of model used, data and control flows, syntactical accuracy, and much more. Due to this, there are several kinds of static analysis methods, which are listed below.

Control flow analysis: This examines the control structures (sequence, selection, and repetition) used in the code. It identifies incorrect and inefficient constructs and also reports unreachable code, that is, the code to which the control never reaches.

Data analysis: This ensures that-proper operations are applied to data objects (for example, data structures and linked lists). In addition, this method also ensures that the defined data is properly used. Data analysis comprises two methods, namely, data dependency and data-flow analysis. **Data dependency** (which determines the dependency of one variable on another) is essential for assessing the accuracy of synchronization across multiple processors. **Dataflow analysis** checks the definition and references of variables.

Fault/failure analysis: This analyzes the fault (incorrect model component) and failure (incorrect behaviour of a model component) in the model. This method uses input-output transformation descriptions to identify the conditions that are the cause for the failure. To determine the failures in certain conditions, the model design specification is checked.

Interface analysis: This verifies and validates the interactive and distributive simulations to check the software code. There are two basic techniques for the interface analysis, namely, model interface analysis and user interface analysis. **Model interface analysis** examines the sub-model interfaces and determines the accuracy of the interface

structure. **User interface analysis** examines the user interface model and checks for precautionary steps taken to prevent errors during the user's interaction with the model'.

c)   **Symbolic Executor:** Symbolic execution concentrates on assessing the accuracy of the model by using symbolic values instead of actual data values for input. Symbolic execution, also known as **symbolic evaluation,** is performed by providing symbolic inputs, which produce expressions for the output.

Symbolic execution uses a standard mathematical technique for representing the arbitrary program inputs (variables) in the form of symbols. To perform the calculation, a machine is employed to perform algebraic manipulation on the symbolic expressions. These expressions include symbolic data meant for execution. The symbolic execution is represented as a symbolic state symbol consisting of variable symbolic values, path, and the path conditions. The symbolic state for each step in the arbitrary input is updated. The steps that are commonly followed for updating the symbolic state considering all possible paths are listed below.

  ✓   The read or the input symbol is created.

  ✓   The assignment creates a symbolic value expression.

  ✓   The conditions in symbolic state add constraints to the path condition.

The output of symbolic execution is represented in the form of a symbolic execution tree. The branches of the tree represent the paths of the model. There is a decision point to represent the nodes of the tree. This node is labeled with the symbolic values of the data at that junction. The leaves of the tree are complete paths through the model and they represent the output of symbolic execution. Symbolic execution helps in showing the correctness of the paths for all computations. Note that in this method the symbolic execution tree increases in size and creates complexity with growth in the model.

d)   **Code Inspection and Reviews**: This technique is a formal and systematic examination of the source code to detect errors. During this process, the software is presented to the project managers and the users for a comment of approval. Before providing any comment, the

inspection team checks the source code for errors. Generally, this team consists of the following.

- **Moderator:** Conducts inspection meetings, checks errors-, and ensures that the inspection process is followed.

- **Reader:** Paraphrases the operation of the software code.

- **Recorder:** Keeps record of each error in the software code. This frees the task of other team members to think deeply about the software code.

- **Author:** Observes the code inspection process silently and helps only when explicitly required. The role of the author is to understand the errors found in the software code.

As mentioned above, the reader paraphrases the meaning of small sections of code during the code inspection process. In other words, the reader translates the sections of code from a compute**r** language to a commonly spoken language (such as English). The inspection process is carried out to check whether the implementation of the software code is done according to the user requirements. Generally, to conduct code inspections the following steps are performed.

a) **Planning:** After the code is compiled and there are no more errors and warning messages in the software code, the author submits the findings to the moderator who is responsible for forming the inspection team. After the inspection team is formed, the moderator distributes the listings as well as other related documents like design documentation to each team member. The moderator plans the inspection meetings and coordinates with the team members.

b) **Overview:** This is an optional step and is required only when the inspection team members are not aware of the functioning of the project. To familiarize the team members, the author provides details to make them understand the code.

c) **Preparation:** Each inspection team member individually examines the code and its related materials. They use a checklist to ensure that each problem area is checked. Each inspection team member keeps a copy of this checklist, in which all the problematic areas are mentioned.

d) Inspection meeting: This is carried out with all team members to review the software code. The moderator discusses the code under review with the inspection team members.

There are two checklists for recording the result of the code inspection, namely, code inspection checklist and inspection error list. **The code inspection checklist** contains a summary of all the errors of different types found in the software code. This checklist is used to understand the effectiveness of inspection process. **The inspection error list** provides the details of each error that requires rework. Note that this list contains details only of those errors that require the whole coding process to be repeated.

All errors in the checklist are classified as major or minor. An error is said to be major if it results in problems and later comes to the knowledge of the user. On the other hand, minor errors are spelling errors and non-compliance with standards. The classification of errors is useful when the software is to be delivered to the user and there is little time to review all the errors present in the software code.

At the conclusion of the inspection meeting, it is decided whether the code should be accepted in the current form or sent back for rework. In case the software code needs reworking, the author makes all the suggested corrections and then compiles the code. When the code becomes error-free, it is sent back to the moderator. The moderator checks the code that has been reworked. If the moderator is completely satisfied with the software code, inspection becomes formally complete and the process of testing the software code begins.

CHECK YOUR PROGRESS

5.   State true or false

a.   The main purpose of code inspection is to find defects and it can also spot process improvement if any.

b.   Inspection is an indiscipline practice for correcting defects in software artifacts.

c.   Code Walkthrough is a form of peer review in which a programmer leads the review process and the other team members ask

---

questions and spot possible errors against development standards and other issues.

1.5   CODING TOOLS

While writing software code, several coding tools are used along with the programming language to simplify the tasks of writing the software code. Note that coding tools vary from one programming language to another as they are developed according to a particular programming language. However, sometimes a single coding tool can be used in more than one programming language. Generally, coding tools comprises text editors, supporting tools for a specific programming language, and the framework required to run the software code .In addition to the programming language and coding tools, there are some software programs that are essential to run the software code. For instance, a **debugger** is used to detect the source of program errors by performing a step-by-step execution of the software code. A debugger breaks program execution at various levels in the application program. It supports features such as breakpoints, displaying or changing memory, and so on. Similarly, **compilers** are used to translate programs written in a high-level language into their machine language equivalents.

1.6   CODING DOCUMENTATION

Code documentation is a manual-cum-guide that helps in understanding and correctly utilizing the software code. The coding standards and naming conventions written in a commonly spoken language in code documentation provide enhanced clarity for the designer. Moreover, they act as a guide for the software maintenance team (this team focuses on maintaining software by improving and enhancing the software after it has been delivered to the end user) while the software maintenance process is carried out. In this way, code documentation facilitates code reusability.

While writing a software code, the developer needs proper documentation for reference purposes. Programming is an ongoing

process and requires modifications from time to time. When a number of software developers are writing the code for the same software, complexity increases. With the help of documentation, software developers can reduce the complexity by referencing the code documentation. Some of the documenting techniques are comments, visual appearances of codes, and programming tools. **Comments** are used to make the reader understand the logic of a particular code segment. The **visual appearance of a code** is the way in which the program should be formatted to increase readability. The **programming tools** in code documentation are algorithms, flowcharts, and pseudo-codes.

Code documentation contains source code, which is useful for the software developers in writing the software code. The code documents can be created with the help of various coding tools that are used to auto-generate the code documents. In other words, these documents extract comments from the source code and create a reference manual in the form of text or HTML file. The auto-generated code helps the software developers to extract the source code from the comments. This documentation also contains application programming interfaces, data structures, and algorithms. There are two kinds of code documentation, namely, internal documentation and external documentation.

1.6.1 Internal Documentation

Documentation which focuses on the information that is used to determine the software code is known as internal documentation. It describes the data structures, algorithms, and control flow in the programs. There are various guidelines for making the documentation easily understandable to the reader. Some of the general conventions to be used at the time of internal documentation are header comment blocks, program comments, and formatting. Header comment blocks are useful in identifying the purpose of the code along with details such as how the c0ge functions and how each segment of code is used in the program.

Since software code is updated and revised several times, it is important to keep a record of the code information so that internal documentation reflects the changes made to the software code. Internal documentation

should explain how each code section relates to user requirements in the software. Generally, internal documentation comprises the following information.

- Name, type, and purpose of each variable and data structure used in the code

- Brief description of algorithms, logic, and error-handling techniques

- Information about the required input and expected output of the program

- Assistance on how to test the software

- Information on the upgradations and enhancements in the program.

1.6.2 External Documentation

Documentation which focuses on general description of the software code and is not concerned with its detail is known as external documentation. It includes information such as function of code, name of the software developer who has written the code, algorithms used in the software code, dependency of code on programs and libraries, and format of the output produced by the software code. Generally, external documentation includes structure charts for providing an outline of the program and describing the design of the program.

External documentation is useful for software developers as it consists of information such as description of the problem along with the program written to solve it. In addition, it describes the approach used to solve the problem, operational requirements of the program, and user interface components. For the purpose of readability and proper understanding, the detailed description is accompanied by figures and illustrations that show how one component is related to another.

External documentation explains why a particular solution is chosen and implemented in the software. It also includes formulas, conditions, and references from where the algorithms or documentation are derived. External documentation makes the user aware of the errors that occur while running the software code. For example, if an array of five

numbers is used, it should be mentioned in the external documentation that the limit of the array is five.

CHECK YOUR PROGRESS

6.   Code documentation contains …………… which is useful for the software developers in writing the software code.

7.   The coding standards and naming conventions written in a commonly spoken language in code documentation provide …….. for the designer.

8.   The **……………** contains a summary of all the errors of different types found in the software code.

9.   Define a debugger.

10.   Define a compiler.

1.7   SUMMING UP

- The most software industries have their own coding standards and they expect that there engineers to adhere to them. On the other hand, coding guidelines serve as general suggestions to programmers regarding good programming styles, but the implementation is left the software engineers.

- Code review is an efficient method of removing errors because of its capability to identify errors. The two main approach of code review are code walkthrough and code inspection.

- Both of these methods are used to identify error at an early stage.

- The important basic programming errors are also discussed and identified in the unit. Code verification is the process used for checking the software code for errors introduced in the coding phase.

- The objective of code verification process is to check the software code in all aspects. This process includes checking the consistency of user requirements with the design phase.

- The code verification techniques are classified into two categories, namely, dynamic and static. The dynamic technique is performed by executing some test data. The outputs of the program are tested to find errors in the software code. This technique follows the conventional approach for testing the software code. In the static technique, the program is executed conceptually and without any data.

- While writing software code, several coding tools are used along with the programming language to simplify the tasks of writing the software code. Note that coding tools vary from one programming language to another as they are developed according to a particular programming language.

- **Code documentation** is a manual-cum-guide that helps in understanding and correctly utilizing the software code. The coding standards and naming conventions written in a commonly spoken language in code documentation provide enhanced clarity for the designer. The internal and external documentation have its own merits and limitations.

1.8   ANSWERS TO CHECK YOUR PROGRESS

1. Programming

2. Code reuse, errors

3. True

4. False

5. a. True, b. False, c. True

6. Source code

7. Enhance clarity

8. Code inspection list

9. It is a debugging tool is a computer program that is used to test and debug other programs.

10. A compiler is a computer program that converts the program written in one programing language (source language) into another programming language (target language)

1.9   POSSIBLE QUESTIONS

Short answer type questions.

1) What are some of the important errors checked during the code walkthrough?

2) Define code walkthrough.

3) Define code inspection.

4) What is internal and external documentation?

5) State some of the important coding Guidelines.

Long answer type questions.

1) When during the development process is the compliance with coding standard is checked?

2) List two coding standards each for:

   i) Enhancing readability of the code.

   ii) Reuse of the code

3) Briefly highlight the difference between code inspection and code walkthrough. Compare the relative merits of code inspection and code walkthrough.

4) What do you understand by static and dynamic analysis of programs? How are static and dynamic program analysis results useful?

5) What is a coding standard? What problems may arise if the engineers of an organization do no adhere to any coding standard?

6) What is the difference between coding standard and coding guideline?

7) Discuss different types of code reviews. Why it is considered to be the most efficient way to remove errors from the code?

8) What is meant by code walkthrough? What are some of the important errors that can be checked during code walkthrough?

9) Define control flow and data analysis.

10) What are the major steps to conduct code Inspection?

1.10 REFERENCES AND SUGGESTED READINGS

- "Fundamentals of Software Engineering", Rajib Mall, Prentice-Hall of India.

- "An Integrated Approach to Software Engineering", Pankaj Jalote, Narosa Publishing House

- http://www.tutorialspoint.com

- http://www.geeksforgeeks.com

UNIT 2: SOFTWARE TESTING I

2.1   INTRODUCTION

The aim of program testing is to identify all defects in a program. However, in practice even after satisfactory completion of the testing phase, it is not possible to guarantee that a program is error free. This is because the input data domain for most of the program is very large and it is not practically possible to test all possible to test all the data exhaustively with respect to each value that a input can assume. Even with this obvious limitation of the testing process, we should not estimate the importance of testing. It is very important to understand that by careful testing we can eliminate or reduce the major defects form the system.

Why to Learn Software Testing?

In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**.

In most cases, the following professionals are involved in testing a system within their respective capacities –

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

Applications of Software Testing

- **Cost Effective Development** - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.

- **Product Improvement** - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.

- **Test Automation** - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.

- **Quality Check** - Software testing helps in determining following set of properties of any software such as

  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

2.2 UNIT OBJECTIVES

After going through the unit you will be able:

- The importance of testing and its applications.
- The basic concepts and terminologies used in the process of testing.
- Designing of test cases and test suites and why we design test cases?
- The main strategic issues related to the testing.
- The different type of testing like unit testing, Integration testing and system testing.

2.3 TESTING FUNDAMENTALS

In this section we will define some of the terms that are commonly used when we discuss about testing. Then we will discuss about some of the basic issues related to how testing can be performed and some desirable properties for the criteria used for testing.

2.3.1 Basic Concepts and Terminologies

The term error and failure used to refer to problems in requirements, design or code. Sometimes error, fault and failure are used interchangeably and sometimes they refer to different concepts. The following are some of the commonly used terms associated with testing:

a) **Error:** It is the mistake committed by development team during any of the development phase. The mistake might have been committed in the requirements, design or code. It is sometimes referred as fault, bug or defect.

b) **Failure:** It is a manifestation of an error. In other words a failure is the symptom of an error. But mere presence of an error may not necessarily lead to failure.

c) **Test Case:** It is the triplet [I , S, O], where I is the data input to the system, S is the state of the system at which the data is input and O is the expected output of the system.

d) **Test Suite:** It is the set of all test cases with which a given software product is tested.

e) **Verification and Validation:** Verification is a process of determining whether the output of one phase of software development conforms to that of its previous phase, whereas validation is the process of determining whether a fully developed system conforms to its requirement specification. Thus the main difference between is that the verification is concerned with phase containment of error and the aim of validation is to make the final product error free.

2.3.2 Test Plan Activities During Testing

Testing involves performing the following main activities:

a) Test suite design: Test suite is a container that has a set of tests which helps testers in executing and reporting the test execution status. It can take any of the three states namely active, in progress and completed.

A Test case can be added to multiple test suites and test plans. After creating a test plan, test suites are created which in turn can have any number of tests.

b) Running test cases and checking the results to detect failure: Each test case is run and the results are compared with the expected results. A mismatch between the actual and expected result indicates a failure. A test cases for which the system fails are noted down for later debugging.

c) Debugging: Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system. For each activity observed during the previous activity, debugging is carried out to identify statements that are in error. Here, the symptoms are analyzed to locate the error.

d) Error Correction: After the error is located in the previous activity the code is appropriately changed to correct the errors.

2.4   TEST CASES AND TEST CRITERIA

Test criteria help **the tester to organize the test process**. They should be chosen in accordance with the available test effort. Test coverage measures are defined as a ratio between the test cases required for satisfying the criteria and those of these which have been executed. A good test should have the following criteria's

(1)   Validity, (2) Reliability, (3) Level of difficulty, (4) Discrimination Power, and (5) The Quality of Options.

2.4.1 Why Design Test Cases?

It is not necessary that if we design test cases then it would not be sufficient to test software using large number of random inputs values? But this technique would be very costly and very ineffective way of testing. In other words, testing a system using a large collection of test cases that are selected at random does not guarantee that all (or even most) of the errors in the system will be uncovered. Consider one example, let there is code segment that determines the greater of two integer values x and y. The code segment gives the simple programming error:

> if (x>y) max = x;
>
> else max = x;

In this segment of code if put values like {(x =3, y=2); (x=2,y=3)} can detect error, whereas a large test suite { (x =3 , y =2), (x = 4 , y =3), (x=5 , y =1)}does not detect the errors. It means that for effective testing we need to design the test suite carefully rather than taking large number of input values.

2.4.2 Approaches to Design Test Cases

A minimal test suite is a carefully designed set of test cases such that each test case helps in detecting different errors. This is in contrast to testing

using some random input values. There are essentially two main approaches to systematically design test cases.

a) Black Box Approach: Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.

This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing. There are different techniques involved in Black Box testing but the following are the two main approaches.

Equivalence Class partitioning: It divides input domain into classes of data, and with the help of these classes of data, test cases can be derived. An ideal test case identifies class of error that might require many arbitrary test cases to be executed before general error is observed. The strategy for black box testing is intuitive and simple and the most important test is the identification of equivalence class.

In equivalence portioning, equivalence classes are evaluated for given input conditions. Whenever any input is given, then type of input condition is checked, then for this input conditions, Equivalence class represents or describes set of valid or invalid states.

Guidelines for Equivalence Partitioning:

- If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.

- If a specific value is given as input, then one valid and two invalid equivalence classes are defined.

- If a member of set is given as an input, then one valid and one invalid equivalence class is defined.

- If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined.

Summary of the Black Box Test Suite Design Approach

➢ Examine the input and output values of the program.

➢ Identify the equivalence classes.

➢ Design equivalence class test cases by picking one representative value form each equivalence class.

➢ Design the boundary values test cases as follows. Examine if any equivalence class is a range of values. Include the values at the boundaries of such equivalence classes in the test suite.

Boundary Value Analysis: Boundary value analysis is a type of black box or specification based testing technique in which tests are performed using the boundary values. Boundary values are validated against both the valid boundaries and invalid boundaries. The boundary value analysis involves designing test cases using the values at the boundaries of different equivalence classes.

b) White Box Approach: White box testing techniques analyse the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. It examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

Testing Techniques:

a) Statement Coverage - This technique is aimed at exercising all programming statements with minimal tests. It aims to design test cases so as to execute every statement in a program at least once. The principal idea gfis that unless a statement is executed there is no way to determine whether an error exists in that statement. The statement coverage is a very intuitive and appealing testing technique.

b) Branch Coverage - This technique is running a series of tests to ensure that all branches are tested at least once. The test cases are designed so as to make each branch condition in the program to assume true or false values in turn. It is also known as edge testing since in this testing scheme each edge of the program's control graph is traversed at least once. It is obviously stronger than statement coverage based testing.

c) Path Coverage - This technique corresponds to testing all possible paths which means that each statement and branch is covered. It requires designing test cases such as all linearly independent paths in the program

are executed at least once. A linearly independent path is defined in terms of control flow graph (CFG) of a program.

Control flow graph: It describes the sequence in which the different instructions of a program gets executed. In other words we can also describe how the control flows through the program.

Advantages of White Box Testing:

- Forces test developer to reason carefully about implementation.

- Reveals errors in "hidden" code.

- Spots the Dead Code or other issues with respect to best programming practices.

Disadvantages of White Box Testing:

- Expensive as one has to spend both time and money to perform white box testing.

- Every possibility that few lines of code are missed accidentally.

- In-depth knowledge about the programming language is necessary to perform white box testing.

CHECK YOUR PROGRESS

1. State true and false.

a. Error and failure are synonymous in software testing terminologies.

b. The main purpose of integration testing is to find design errors.

2. The ….is the set of all test cases with which a given software product is tested.

3. ………….and ………… are the two main approaches to systematically design test cases.

2.5   STRATEGIC ISSUES IN TESTING

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

Following are the issues considered to implement software testing strategies.

- Specify the requirement before testing starts in a quantifiable manner.

- According to the categories of the user generate profiles for each category of user.

- Produce a robust software and it's designed to test itself.

- Should use the Formal Technical Reviews (FTR) for the effective testing.

- To access the test strategy and test cases FTR should be conducted.

- To improve the quality level of testing generate test plans from the user's feedback.

2.6 UNIT TESTING

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality. Before carrying out unit test cases have to be designed and the test environment for the unit under test has to be developed. In the next section we will discuss the environment needed to perform unit testing.

Driver and Stub Modules: In order to test a single module, we need a complete environment to provide all relevant code that is necessary for execution of the module. Other than the module under test the following are needed to test the module:

- The procedures belonging to other modules that the module under test calls.

- Non local data structures that the module accesses.

- A procedure to call the functions of the module under test with appropriate parameters.

The stubs and drivers are designed to provide the complete environment for a module so that testing can be carried out.

- **Stubs:** Stubs are developed by software developers to use them in place of modules, if the respective modules aren't developed, missing in developing stage, or are unavailable currently while Top-down testing of modules. A Stub simulates module which has all the capabilities of the unavailable module. Stubs are used when the lower-level modules are needed but are unavailable currently.

- **Drivers:** Drivers serve the same purpose as stubs, but drivers are used in Bottom-up integration testing and are also more complex than stubs. Drivers are also used when some modules are missing and unavailable at time of testing of a specific module because of some unavoidable reasons, to act in absence of required module. Drivers are used when high-level modules are missing and can also be used when lower-level modules are missing.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Difference between Stubs and Drivers:

S.No.	Stubs	Drivers
1.	Stubs are used in Top-Down Integration Testing.	Drivers are used in Bottom-Up Integration Testing.

S.No.	Stubs	Drivers
2.	Stubs are basically known as a "called programs" and are used in the Top-down integration testing.	While, drivers are the "calling program" and are used in bottom-up integration testing.
3.	Stubs are similar to the modules of the software that are under development process.	While drivers are used to invoking the component that needs to be tested.
4.	Stubs are basically used in the unavailability of low-level modules.	While drivers are mainly used in place of high-level modules and in some situation as well as for low-level modules.
5.	Stubs are taken into use to test the feature and functionality of the modules.	Whereas the drivers are used if the main module of the software isn't developed for testing.
6.	The stubs are taken into concern if testing of upper-levels of the modules are done and the lower-levels of the modules are under developing process.	The drivers are taken into concern if testing of lower-levels of the modules are done and the upper-levels of the modules are under developing process.
	Stubs are used when lower-level of modules are missing or in a partially developed phase, and we want to test the main module.	Drivers are used when higher-level of modules are missing or in a partially developed phase, and we want to test the lower (sub) - module.

CHECK YOUR PPROGRESS

4. **State true or false:**

   a. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

   b. The stubs and drivers are not designed to provide the complete environment for a module so that testing can be carried out.

5. The ……… strategy aims to design test cases so as to execute every statement in a program at least once.

6. The ………… describes the sequence in which the different instructions of a program get executed.

2.7 INTEGRATION TESTING

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

S.No.	Integration Testing Method
1	**Bottom-up integration** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-down integration** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

Integration test approaches –

There are four types of integration testing approaches. Those approaches are the following:

a) Big-Bang Integration Testing –
It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing are very expensive to fix.

Advantages:

- It is convenient for small systems.

Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.

- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

b) Bottom up Integration Testing: In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

Advantages:

- In bottom-up testing, no stubs are required.

- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

- Driver modules must be produced.

- In this testing, the complexity that occurs when the system is made up of a large number of small subsystem.

c) Top-Down Integration Testing –

Top-down integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

- Separately debugged module.

- Few or no drivers needed.

- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.

- Modules at lower level are tested inadequately.

2.8 ACCEPTANCE TESTING

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application. By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

a) Alpha Testing:

Alpha testing is used to determine the product in the development testing environment by a specialized tester's team usually called alpha testers.

b) Beta Testing:

Beta testing is used to assess the product by exposing it to the real end-users, usually called beta testers in their environment. Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

Use of Acceptance Testing:

- To find the defects missed during the functional testing phase.

- How well the product is developed.

- A product is what actually the customers need.

- Feedbacks help in improving the product performance and user experience.

- Minimize or eliminate the issues arising from the production.

CHECK YOUR PROGRESS

7. **State true or false**

   a. The big bang approach is preferred for integration testing of large programs.

   b. Equivalence class partitioning is a white box testing strategy.

   c. Testing cannot catch each and every bug in an application

8. Integration testing can be done in two ways ………….. and …………

9. The ….. and …. are designed to provide the complete environment for a module so that testing can be carried out.

10.   A linearly independent path is defined in terms of …………. of a program.

2.9   SUMMING UP

- In this unit we mainly discussed about the concept and different approaches to testing.

- Exhaustive testing of almost any system is almost impractical. Also random selection of test cases is inefficient since many test cases become redundant as they detect the same type of errors.

- There are mainly two main approaches to testing: Black box testing and white box testing. Designing test cases for black box testing does not require any knowledge about how the function have been designed and implemented. On the other hand white box testing requires knowledge about internals of the software.

- Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

- Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

- Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

2.10 ANSWERS TO CHECK YOUR PROGRESS

1. a. True, b. True

2. Test suites

3. Black box and white box

4. a. True, b. False

5. Statement coverage based

6. Control flow graph

7. a. True, b. False, c. True

8. Bottom – up and top down

9. Stubs and drivers

10. Control flow graph

2.11 POSSIBLE QUESTIONS

Short answer type questions.

1) What are driver and stub modules? Why are they required?

2) Distinguish between error and failure in the context of program testing.

3) Distinguish between software verification and validation.

4) Define acceptance testing.

5) Define control flow graph.

6) List any three differences between drivers and stubs?

7) What is Big Bang Integration testing?

8) What are the advantages of top down integration testing?

9) Define statement and branch coverage in testing. Which testing approach is stronger?

10) List the main activities to perform testing in software development.

Long answer type questions.

1) What are different approaches of integration testing? Which approach is more preferred for the large projects?

2) What is difference between white box and black box testing?

3) What do you mean by equivalence class partitioning and boundary value analysis?

4) Prove that the branch coverage based testing is a stronger testing technique compared to a statement coverage based testing.

5) What do you mean by big bang integration testing? How it is performed?

6) Explain the difference between testing in large and testing in small. What is the purpose of each?

7) What are alpha, beta and acceptance testing? What are the differences among these different types of software product?

8) Define test cases and test suites. Why do we need to design test cases?

9) What are drivers and stubs? Why they are important? What are the major factors that are required to test the module?

10) What do you understand by the unit testing? Discuss in brief about the statement and branch coverage testing strategy.

2.12 REFERENCES AND SUGGESTED READINGS

- "Fundamentals of Software Engineering", Rajib Mall, Prentice-Hall of India.

- "An Integrated Approach to Software Engineering", Pankaj Jalote, Narosa Publishing House

- http://www.tutorialspoint.com

- http://www.geeksforgeeks.com

UNIT 3: SOFTWARE TESTING II

Unit Structure:

3.1 INTRODUCTION

The goal of testing is to find errors, and a good test is one that has a high probability of finding an error. In this unit, different software testing techniques has been discussed.

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand various concepts of white box testing, black box testing, their advantages and disadvantages.

- Understand the difference white box testing vs black box testing.

- Understand the different techniques used in black box and white box techniques

- Understand the need of Cause Effect Graphing their benefits of Cause Effect

3.3 WHITE BOX TESTING

White-box testing is a testing technique which checks the internal functioning of the system. In this method, testing is based on coverage of code statements, branches, paths or conditions. White-Box testing is considered as low-level testing. It is also called glass box, transparent box, clear box or code base testing. The white-box Testing method assumes that the path of the logic in a unit or program is known. In this technique, the internal structure and implementation of how an application works are known to the tester.

Let's assume that there is a car which is not working and therefore you take it to a mechanic to get it fixed. Now the mechanic will examine why the car is not working. Similarly, a tester studies the code of an application and determines all the inputs, and verifies the outputs against desired outcomes.

We need White box testing for the following two reasons

    1) To systematically derive tests from source code.

    2) To know when to stop testing.

As a tester, when performing white box testing, your goal was clear: to derive classes out of the requirement specifications, and then to derive test cases for each of the classes. You were satisfied once all the classes and boundaries were systematically exercised. First, it gives us a systematic way to devise tests. As we will see, a tester might focus on testing all the lines of a program; or focus on the branches and conditions of the program. Different criteria produce

different test cases. Second, to know when to stop. It is easy to imagine that the number of possible paths in a mildly complex piece of code is just too large, and exhaustive testing is impossible. Therefore, having clear criteria on when to stop helps testers in understanding the costs of their testing.

Following are the steps that are taken into consideration while performing white box testing:

- Verification of security holes in source code

- Testing of any broken or incomplete path

- To verify the flow of structure as mentioned in the software requirement document

- To check the conditionality of all loops and the overall functionality of the software

- To check if all the expected outcomes are met

- Line by line verification of code

The main aim of white box testing is to verify the proper flow and functionality of the application. The test cases are executed and the output is compared to the desired outcome, if any of the output does not match with the expected outcome, it means that a bug is encountered.

3.3.1 White Box Testing Tools

Some of the common white box testing tools used is given below

- Veracode

- RCUNIT

- cfix

- Googletest

- EMMA

- NUnit

3.3.2 White Box Testing Example

For understanding how to create test cases in white box testing, let's consider the pseudo code given below:

```
INPUT A & B

C = A + B

IF C>100

PRINT "ITS DONE"

ELSE

PRINT "ITS PENDING"
```

Since the goal of white box testing is to verify and cross check all the different loops, branches and decision statements, so to exercise white box testing in the code given above, the two test cases would be –

A= 33, B=45

A=40, B=70

For the first test case, A=33, B=45; C becomes 78, due to which it will skip the 4th line in the pseudo code, since C < 100 and will directly print the 6th line, i.e. ITS PENDING.

Now, for the second test case, A=40, B=70; C becomes 110, which means that C > 100 and therefore it will print the 4th line and the program will be stopped.

These test cases will ensure that each line of the code is traversed at least once and will verify for both true and false conditions.

3.3.3 Advantages of White Box Testing

- Optimization of code by revelation of hidden faults.

- Transparency of the internal code structure that helps to derive the type of input data needed to adequately test an application.

- This incorporates all conceivable code paths, enabling a software engineering team to carry out comprehensive application testing.

3.3.4 Disadvantages of White Box Testing

- A complicated and expensive process that involves the skill of an experienced professional, programming ability and knowledge of the underlying code structure.

- A new test script is necessary when the execution changes too frequently.

- Detailed testing with the white box testing approach is significantly more demanding if the application covers many different areas, such as the Gojek Super App.

3.4 CODE COVERAGE

Code coverage is a measure which describes the degree of which the source code of the program has been tested. It is one form of white box testing which finds the areas of the program not exercised by a set of test cases. It also creates some test cases to increase coverage and determining a quantitative measure of code coverage. In most cases, code coverage system gathers information about the running program. It also combines that with source code information to generate a report about the test suite's code coverage. Following are the code coverage methods:

1. **Statement Coverage**

One of the main objectives of white box testing is to cover as much of the source code as possible. Code coverage is a measure that indicates how much of an application's code contains unit tests that validate its functioning.

Using concepts such as statement coverage, branch coverage, and path coverage, it is possible to check how much of an application's logic is really executed and verified by the unit test suite. These different white box testing techniques are explained below.

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statements}} \times 100$$

2. **Branch Coverage**

In programming, "branch" is equivalent to, say, an "IF statement" where True and False are the two branches of an IF statement.

As a result, in Branch coverage, we check if each branch is processed at least once.

There will be two test conditions in the event of an "IF statement":

One is used to validate the "true" branch, while the other is used to validate the "false" branch.

$$\text{Branch Coverage} = \frac{\text{Number of Executed Branches}}{\text{Total Number of Branches}}$$

3. Path Coverage

Path coverage examines all the paths in a given program. This is a thorough strategy that assures that all program paths are explored at least once. Path coverage is more effective than branch coverage. This method is handy for testing complicated applications.

4. Decision Coverage

Decision Coverage is a white box testing methodology that reports the true or false results of each boolean expression present in the source code. The purpose of decision coverage testing is to cover and validate all available source code by guaranteeing that each branch of each potential decision point is traversed at least once.

A decision point is a point when there is a possibility of occurrence of two or more outcomes from control flow statements such as if statement, do while statement or a switch case statement.

Expressions in this coverage can become difficult at times. As a result, achieving 100% coverage is quite difficult.

$$\text{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}}$$

5. Condition Coverage

Condition coverage, also known as expression coverage, is a testing method for testing and evaluating the variables or sub-expressions in a conditional statement. The purpose of condition coverage is to examine the outcome of each logical condition.

Only expressions with logical operands (an operand is considered as a logical operand if it has its output as either TRUE or FALSE) are examined in this coverage. Condition coverage does not ensure complete decision coverage.

6. Multiple Condition Coverage

In this testing technique, all the different combinations of conditions for each decision are evaluated. For example, we have the following expression,

if (A||B)

then

Print C

So, in this case, the test cases would be as given below

TEST CASE1: A=TRUE, B=TRUE

TEST CASE2: A=TRUE, B=FALSE

TEST CASE3: A=FALSE, B=TRUE

TEST CASE4: A=FALSE, B=FALSE

The point to be noted here is that in this example we have 2 expressions A and B, and as result we have 4 test cases. So, similarly, for 3 conditions we will have 8 test cases.

So, the general formula for Multiple Condition Coverage is that for n conditions, there will be 2n test cases.

7. **Finite State Machine Coverage**

Finite state machine coverage is one of the most difficult forms of code coverage approach. This is due to the fact that it works on the design's functionality. This coverage approach requires you to count the number of times a state is visited or transited. It also determines how many sequences are contained within a finite state system. A sequence in a Finite State Machine is a sorted list of inputs or outputs.

3.5 DATA FLOW TESTING

Data Flow Testing is the test technique that focuses on data variables and their values, which are utilized by using the control flow diagram for the programming logic of the software product. In data flow testing every data variable is tracked and verified. The primary principle behind this test is to identify coding problems that might result in incorrect implementation and use of data variables or data values, i.e. data anomalies like variables declared but not used in the code later, in the software code.

There are two types of data flow testing:

Static data flow testing: The declaration, usage, and deletion of the variables are examined without executing the code. A control flow graph is helpful in this.

Dynamic data flow testing: The variables and data flow are examined with the execution of the code.

Advantages of data flow testing

Data flow testing helps catch different kinds of anomalies in the code. These anomalies include:

- Using a variable without declaration

- Deleting a variable without declaration

- Defining a variable two times

- Deleting a variable without using it in the code

- Deleting a variable twice

- Using a variable after deleting it

- Not using a variable after defining it

Disadvantages of data flow testing

A few disadvantages of data flow testing are:

- Good knowledge of programming is required for proper testing

- Expensive

- Time consuming

Techniques of data flow testing

Data flow testing can be done using one of the following two techniques:

- Control flow graph

- Making associations between data definition and usages

Control flow graph: A control flow graph is a graphical representation of the flow of control, i.e., the order of statements in which they will be executed.

Making associations: In this technique, we make associations between two kinds of statements:

- Where variables are defined

- Where those variables are used

3.6 LOOP TESTING

Loop Testing is a kind of software testing that focuses exclusively on the correctness of loop constructions. It is a component of Control Structure Testing (path testing, data validation testing, condition testing). Loop testing is an example of white-box testing. This approach is used to test software loops.

The following are some examples of loop tests −

- Simple loop
- Nested loop
- Concatenated loop
- Unstructured loop

Need of Loop Testing:

Following are some of the reasons why loop testing is performed −

- Testing can help to resolve loop recurrence concerns.
- Loop testing can indicate constraints in efficiency and operations.
- The loop's uninitialized variables can be identified by testing loops.
- It aids in the identification of loop initiation issues.

Complete Methodology of Loop Testing:

It must be tested at three separate stages within the testing loop −

- When the loop is activated.
- When the loop is executed.
- When the loop is terminated.

The following is the testing technique for all of these loops −

Simple Loop

The following is how a simple loop is tested −

- Ignore the entire loop.
- Make a single pass across the loop.

- Make a number of passes through the loop where a<b, n is the maximum limit of passes.

- Make b, b-1; b+1 passes through the loop, where "b" is the highest amount of passes through the loop allowed.

1. **Nested Loop**

Following steps must be performing to create a nested loop −

- Adjust all the other loops to their smallest value and begin with the innermost loop.

- Initiate a simple loop test on the innermost loop and keep the outside loops at their smallest iteration parameter value.

- Conduct the test for the following loop and make your way outwards.

- Keep testing till the outermost loop is reached.

2. **Concatenated Loops**

If two loops in a chained loop are free of one other, they are checked as simple loops; otherwise, they are tested as nested loops. However, if the loop counter for one loop is utilized as the starting value for the others, the loops are no longer considered separate.

3. **Unstructured Loops**

For unstructured loops, the architecture must be restructured to represent the use of structured programming techniques.

Limitation in Loop testing

- Loop issues are especially common in low-level applications.

- The flaws discovered during loop testing are not significant.

- Numerous defects may be identified by the operating system, resulting in storage boundary breaches, identifiable pointer failures, and so on.

3.7 BLACK BOX TESTING

Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an

input, and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

3.7.1 Types of Black Box Testing

Black box testing can be applied to three main types of tests: functional, non-functional, and regression testing.

1. Functional Testing

Black box testing can test specific functions or features of the software under test. For example, checking that it is possible to log in using correct user credentials, and not possible to log in using wrong credentials.

Functional testing can focus on the most critical aspects of the software (smoke testing/sanity testing), on integration between key components (integration testing), or on the system as a whole (system testing).

2. Non-Functional Testing

Black box testing can check additional aspects of the software, beyond features and functionality. A non-functional test does not check "if" the software can perform a specific action but "how" it performs that action.

Black box tests can uncover if software is:

- Usable and easy to understand for its users
- Performant under expected or peak loads
- Compatible with relevant devices, screen sizes, browsers or operating systems
- Exposed to security vulnerabilities or common security threats

3. Regression Testing

Black box testing can be used to check if a new version of the software exhibits a regression, or degradation in capabilities, from one version to the next. Regression testing can be applied to functional aspects of the software (for example, a specific feature no longer works as expected in the new version), or non-functional

aspects (for example, an operation that performed well is very slow in the new version).

3.7.2 Black Box Testing Techniques

1. Equivalence Partitioning

Equivalence Partitioning or Equivalence Class Partitioning is type of black box testing technique which can be applied to all levels of software testing like unit, integration, system, etc. In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduce time required for testing because of small number of test cases.

It divides the input data of software into different equivalence data classes. Testers can divide possible inputs into groups or "partitions", and test only one example input from each group. For example, if a system requires a user's birth date and provides the same response for all users under the age of 18, and a different response for users over 18, it is sufficient for testers to check one birth date in the "under 18" group and one date in the "over 18" group.

EQUIVALENCE PARTITIONING has been categorized into two parts:

- Pressman Rule.
- Practice Method.

a. Pressman Rule:

Rule 1: If input is a range of values, then design test cases for one valid and two invalid values.

Amount: [_____]   100 − 5000 range of values

For example:
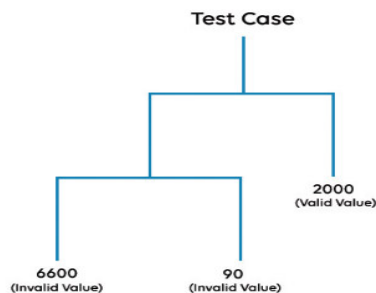
Rule 2: If input is a set of values, then design test cases for all valid value sets and two invalid values.

For example:

Consider any online shopping website, where every product should have a specific product ID and name. Users can search either by using name of the product or by the product ID. Here, you can consider a set of products with product IDs and you want to check for Laptops (valid value).
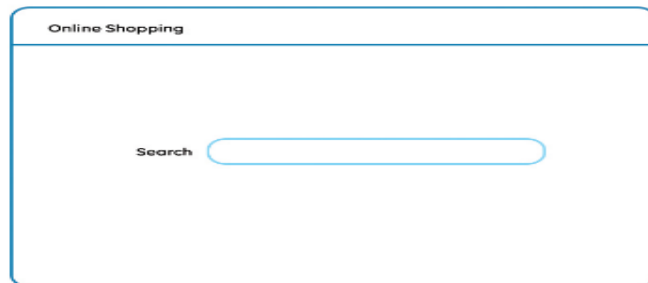
Requirement says:

Product	Product ID
Mobiles	55
Laptops	61
Pen Drives	64
Keyboard	71
Headphones	76
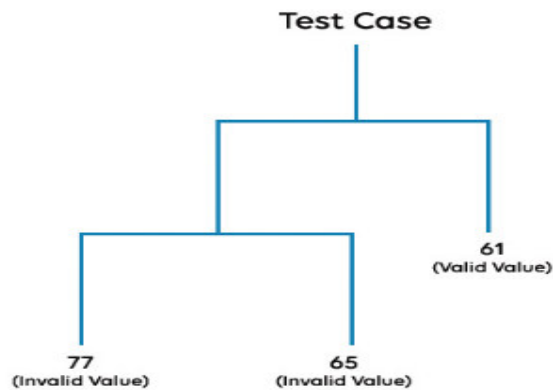
Rule 3: If input is Boolean, then design test cases for both true and false values.

For example:

Consider a sample web page which consists of first name, last name, and email text fields with radio buttons for gender which use Boolean inputs.

If the user clicks on any of the radio buttons, the corresponding value should be set as the input. If the user clicks on a different option, the value of input needs to be updated with the new one (and the previously selected option should be deselected).

Here, the instance of a radio button option being clicked can be treated as TRUE and the instance where none are clicked, as FALSE. Also, two radio buttons should not get selected simultaneously; if so, and then it is considered as a bug.

b. **Practice Method:**

If the input is a range of values, then divide the range into equivalent parts. Then test for all the valid values and ensure that 2 invalid values are being tested for.

Note:

- If there is deviation in between the range of values, then use Practice Method.

- If there is no deviation between the ranges of values, then use Pressman Rule.

2. Boundary Value Analysis

BVA is another Black Box Test Design Technique, which is used to find the errors at boundaries of input domain (tests the behavior of a program at the input boundaries) rather than finding those errors in the centre of input. So, the basic idea in boundary value testing is to select input variable values at their: minimum, just above the minimum, just below the minimum, a nominal value, just below the maximum, maximum and just above the maximum. That is, for each range, there are two boundaries, the lower boundary (start of the range) and the upper boundary (end of the range) and the boundaries are the beginning and end of each valid partition. We should design test cases which exercise the program functionality at the boundaries, and with values just inside and outside the boundaries. Boundary value analysis is also a part of stress and negative testing.

Testers can identify that a system has a special response around a specific boundary value. For example, a specific field may accept only values between 0 and 99. Testers can focus on the boundary values (-1, 0, 99 and 100), to see if the system is accepting and rejecting inputs correctly.

Suppose, if the input is a set of values between A and B, then design test cases for A, A+1, A-1 and B, B+1, B-1.

Example:

Age: | Enter Age | *Accepts any value from 18 to 56

Boundary Value Analysis		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max+1)
17	18,19,55,56	57

Boundary value analysis Vs. Equivalence partitioning:

S.N.	Boundary value analysis	Equivalence partitioning
1.	It is a technique where we identify the errors at the boundaries of input data to discover those errors in the input center.	It is a technique where the input data is divided into partitions of valid and invalid values.
2.	Boundary values are those that contain the upper and lower limit of a variable.	In this, the inputs to the software or the application are separated into groups expected to show similar behavior.
3.	Boundary value analysis is testing the boundaries between partitions.	It allows us to divide a set of test conditions into a partition that should be considered the same.
4.	It will help decrease testing time due to a lesser number of test cases from infinite to finite.	The Equivalence partitioning will reduce the number of test cases to a finite list of testable test cases covering maximum possibilities.
5.	The Boundary Value Analysis is often called a part of the Stress and Negative Testing.	The Equivalence partitioning can be suitable for all the software testing levels such as unit, integration, system.
6.	Sometimes the boundary value analysis is also known as Range Checking.	Equivalence partitioning is also known as Equivalence class partitioning.

3.7.3 Cause Effect Graphing

Cause Effect Graphing is a technique in which a graph is used to represent the situations of combinations of input conditions. The

graph is then converted to a decision table to obtain the test cases. Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions. But since there may be some critical behavior to be tested when some combinations of input conditions are considered, that is why cause-effect graphing technique is used.

Steps used in deriving test cases using this technique are:

1. **Division of specification:**
   Since it is difficult to work with cause-effect graphs of large specifications as they are complex, the specifications are divided into small workable pieces and then converted into cause-effect graphs separately.

2. **Identification of cause and effects:** This involves identifying the causes(distinct input conditions) and effects(output conditions) in the specification.

3. **Transforming the specifications into a cause-effect graph:** The causes and effects are linked together using Boolean expressions to obtain a cause-effect graph. Constraints are also added between causes and effects if possible.

4. **Conversion into decision table:** The cause-effect graph is then converted into a limited entry decision table.

5. **Deriving test cases:** Each column of the decision-table is converted into a test case.

Basic Notations used in Cause-effect graph:
Here **c** represents **cause** and **e** represents **effect**.

The following notations are always **used between a cause and an effect**:

1. **Identity Function:** if c is 1, then e is 1. Else e is 0.

2. **NOT Function:** if c is 1, then e is 0. Else e is 1
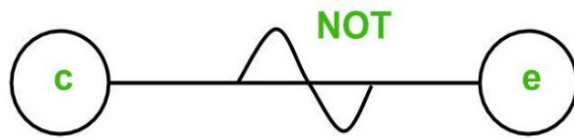
3. **OR Function:** if c1 or c2 or c3 is 1, then e is 1. Else e is 0.
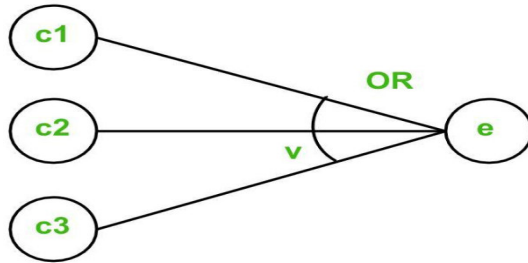
4. **AND Function:** if both c1 and c2 and c3 is 1, then e is 1. Else e is 0
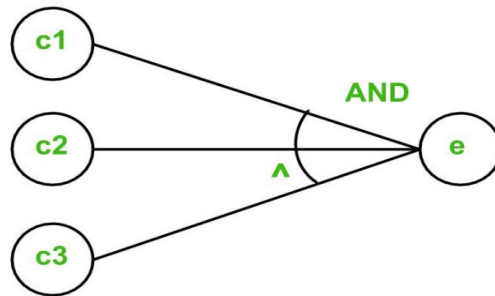
To represent some impossible combinations of causes or impossible combinations of effects, constraints are used. The following **constraints** are used in cause-effect graphs:

a. **Exclusive   constraint** or **E-constraint:** This   constraint exists between causes. It states that either c1 or c2 can be 1, i.e., c1 and c2 cannot be 1 simultaneously.
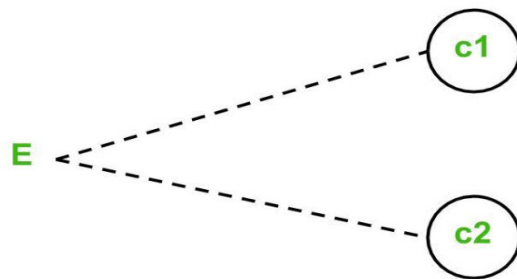
b. **Inclusive constraint** or **I-constraint:** This constraint exists between causes. It states that atleast one of c_1, c_2 and c_3 must always be 1, i.e., c_1, c_2 and c_3 cannot be 0 simultaneously.
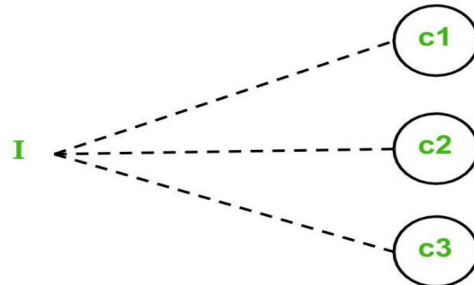
c. **One and Only One constraint** or **O-constraint:** This constraint exists between causes. It states that one and only one of c_1 and c_2 must be 1.
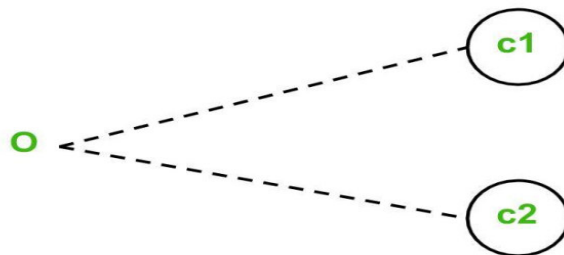
d. **Requires constraint** or **R-constraint:** This constraint exists between causes. It states that for c_1 to be 1, c_2 must be 1. It is impossible for c_1 to be 1 and c_2 to be 0.
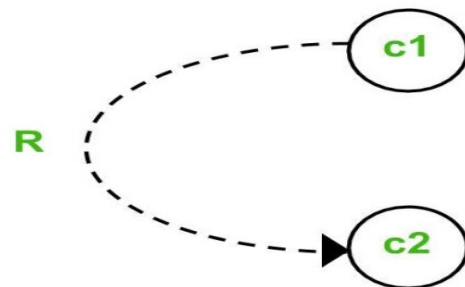
e. **Mask constraint** or **M-constraint:** This constraint exists between effects. It states that if effect e_1 is 1, the effect e_2 is forced to be 0.
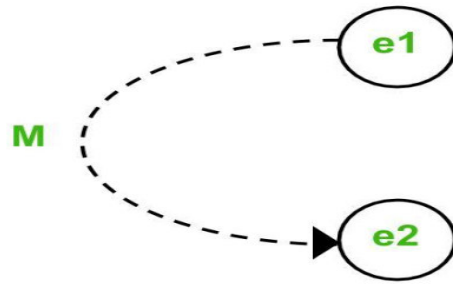
Benefits of Cause Effect Graph Technique

- It helps us to determine the root causes of a problem or quality using a structured approach.

- It uses an orderly, easy-to-read format to diagram cause-and-effect relationships.

- It indicates possible causes of variation in a process.

- It Identifies areas, where data should be collected for further study.

- It Encourages team participation and utilizes the team knowledge of the process.

- It Increases knowledge of the process by helping everyone to learn more about the factors at work and how they relate.

3.8 COMPARISON OF WHITE BOX TESTING AND BLACK BOX TESTING

Black Box Testing	White Box Testing
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.

It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	**Example:** by input to check and verify loops

3.9 MUTATION TESTING

Mutation testing is a structural testing technique, which uses the structure of the code to guide the testing process. On a very high level, it is the process of rewriting the source code in small ways in order to remove the redundancies in the source code

These ambiguities might cause failures in the software if not fixed and can easily pass-through testing phase undetected.

3.9.1 Mutation Testing Benefits

Following benefits are experienced, if mutation testing is adopted:

- It brings a whole new kind of errors to the developer's attention.

- It is the most powerful method to detect hidden defects, which might be impossible to identify using the conventional testing techniques.

- Tools such as Insure++ help us to find defects in the code using the state-of-the-art.

- Increased customer satisfaction index as the product would be less buggy.

- Debugging and maintaining the product would be easier than ever.

3.9.2 Mutation Testing Types

- Value Mutations: An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.

- Decision Mutations: The decisions/conditions are changed to check for the design errors. Typically, one changes the arithmetic operators to locate the defects and also we can consider mutating all relational operators and logical operators (AND, OR , NOT)

- Statement Mutations: Changes done to the statements by deleting or duplicating the line which might arise when a developer is copy pasting the code from somewhere else.

CHECK YOUR PROGRESS

1. Why we need white box testing?
2. What are steps considered during white box testing?
3. Write the advantage and disadvantage of white box testing?
4. What is code coverage? Explain the different code coverage methods?

3.10 SUMMUING UP

- White box testing can be quite complex. The complexity involved has a lot to do with the application being tested. A small application that performs a single simple operation could be white box tested in few minutes, while larger programming applications take days, weeks and even longer to fully test.

- White box testing in software testing should be done on a software application as it is being developed after it is written and again after each modification

- Boundary value analysis is testing at the boundaries between partitions.

- Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same.

- Boundary Value Analysis is better than Equivalence Partitioning as it considers both positive and negative values along with maximum and minimum value. So, when compared with Equivalence Partitioning, Boundary Value Analysis proves to be a better choice in assuring the quality.

- Black Box testing, also known as Behavioral Testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional

3.11 ANSWERS TO CHECK YOUR PROGRESS

1. Equivalence Partitioning: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.

2. Boundary Value Analysis: It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.

3. Cause-Effect Graphing: It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

4. Mutation testing is a white box method in software testing where we insert errors purposely into a program (under test) to verify whether the existing test case can detect the error or not. In this testing, the mutant of the program is created by making some modifications to the original program.

3.12 POSSIBLE QUESTIONS

Short answer type questions:

1. What is data flow testing? Explain their advantages and disadvantages?
2. What is loop testing? Explain its needs?
3. What is black box testing? Explain its different types?

Long answer type questions:

1. Explain different black box testing techniques?
2. Explain the black box testing techniques?
3. Explain Cause Effect Graphing?
4. Explain Mutation testing?

3.13 REFERENCES AND SUGGESTED READINGS

- Mall Rajib, Fundamentals of Software Engineering, PHI.

- Pressman, Software Engineering Practitioner's Approach, TMH.

UNIT 4: SOFTWARE MAINTENANCE

Unit Structure:

4.1 INTRODUCTION

Whenever a software is delivered, there might be some need to make a change. There is always a scope for improvement and that improvement brings change in picture. Changes may be requiring for modify or update any existing solution to create a new solution to a problem. This comes under software maintenance part.

Similar to software development, a maintenance request often goes through a lifecycle. At first the request is analysed, what will be its impact on the system is determined, then any required modifications are designed, coded, tested and finally implemented. Training is a corecomponents of the software maintenance phase. Maintainer need to understand the existing code carefully. Simple code and good documentation at the development stage is more helpful at development stage especially of developer is not available or if there's been a long gap since development.

4.2 UNIT OBJECTIVES

After going through this unit student will be able to learn

- Basic concept of Software maintenance

- Importance of Software maintenance.

- Different types of Software Maintenance.

- Software Maintenance Process.

- Different techniques involve in Software Maintenance.

- Layer Structure of Software Maintenance.

4.3 BASIC CONCEPT AND IMPORTANCE OF SOFTWARE MAINTENANCE

Software maintenance is changes to a product maintenance or service while maintaining its integrity after software has been promoted to production. Software maintenance is a part of software engineering. Software maintenance is also software evaluation based on user's feedback.

Software maintenance last longer than software development. For example, Software development may last from one year but software maintenance may last for 5 to 10 years. This is because organization want to get most return on investment in software development. For the same reason there are jobs in Software management then in software development.

4.3.1 Why Software Maintenance ss Required?

The reason for software maintenance is

1. To fix bugs also called corrective maintenance.

2. To implement enhancements requirement by users or new regulations. Different sources say this is the largest percentage of work in software maintenance.

3. To increase non-functionalities qualities like performance, security, design ,usability of the software.

4. To decrease software complexity . For example by code refactoring or data refactoring.

5. Software maintenance is also required to make it work in a new environment, upgraded hardware for upgraded operating system, new database management system or other software, making the software run in a new environment is also called adaptive maintenance.

6. Finally software maintenance is required to delete retired functionalities.

4.3.2 Software Management Agreement

This agreement states the scope of the Software maintenance. It also states transition, service level management and incident management. Transition means training from the software developers, in setting up esteemed help desk. Service level agreements and incident management means receives request ,log the request, prioritize the request and send them the responsible role and track them until they are closed. Software level agreement also mention software management process.

STOP TO CONSIDER

Maintenance of the product, after deploying of the product is known is Software Maintenance. It is a process of modifying a software system or a component after delivery to correct faults, improve other attributes.

4.4 TYPES OF SOFTWARE MAINTENANCE

In a software lifetime, types of maintenance may vary based on its nature. Based on types of software maintenance is different. 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 4 types of software maintenance based on characteristics:

1. Corrective maintenance
2. Adaptive maintenance
3. Perfective maintenance
4. Preventive maintenance

Let's discuss each one of them.

1. Corrective maintenance:

   This include modification done in order to fix the problems. If after delivered if any bug is reported by user that bug need to corrected. So that type of maintenance comes under corrective maintenance. It deals with the repair of defects found in day-to-day system functions.

2. Adaptive maintenance:

   The major concept is if we have made some modification in some part of software and because of that change ,there is a need to maintain all the part of the software .This include modifications applied to keep the software product upto date. It is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system.

3. Perfective maintenance:

   To keep the software usable over long period of time,whatever modification required that comes under perfective maintenance. To improve its reliability and performance, it includes new features, new user requirements for refining the software . This includes changing the functionalities of as per users changing needs.

4. Preventive maintenance:

   To prevent future problems of software ,whatever modification need to do that comes under preventive maintenance. The problem which are not significant at this moment but they may cause serious issues in future, those kind of problem comes under this. It comprises documentation updating, code optimization and code reconstructing.

STOP TO CONSIDER

In short, corrective maintenance is taking the existing code and correcting the faults that deviates from document requirement. Adaptive maintenance is adding new features to existing code. Perfective maintenance is typically made to improve maintainability of the code. Preventive maintenance is usually made as a result of code inspection to reduce likelihood of a failure.

CHECK YOUR PROGRESS

1. Software maintenance is categorized into how many categories?

4.5 SOFTWARE MAINTENANCE LIFE CYCLE
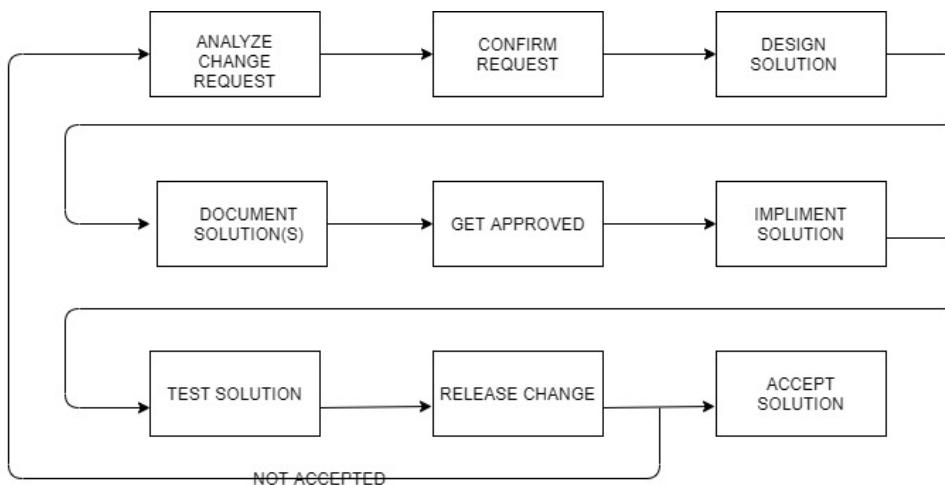
SOFTWARE MAINTENANCE PROCESS

Depending on types of the software being maintained, the maintenance process vary. If it is small means little effort requires and if it is big then more effort required. The most expensive part of software life cycle is software maintenance process. Software maintenance process is done by software maintainer. It performs various task like analyze change request, confirm the request or denied request based on analysis. After that it designs one or more possible solutions. Then user approved one of the solution. After that maintenance developers implement the

solution and then the maintenance tester test the solution and finally is accepted by the user. If solution is not accepted by the user then the process is repeated right from the analyzing change request.

4.6 TECHNIQUES OF SOFTWARE MAINTENANCE

It involves the following techniques:

1. Software Configuration Management

2. Impact Analysis

3. Software Rejuvenation

4.6.1 Software Configuration Management

While developing the software, various documents ,image files,core files, databases, script and different types of entities need to manage during entire software development life cycle. So in general words , software configuration management is to systematically manage, organize and control the changes in the document, codes and other entities. So whatever activities carried out to manage those changes are Software management configuration.

Four primary objectives:

- To find out all items that collectively define the software configuration. Items may be class file, script file, may be project file etc.

- To manage changes to one or more of these items that are obtained during first step.

- To facilitate the construction of different versions of an application. Software engineering is a continuous process. At a time, software is not going to develop. At first, initial version is developed, next the second version and so on. So the

development of the version step by step-l is the main objective of this step.

- To make sure that software quality is maintained as the configuration evolves over time. Changes which are made during different versions are to be such that quality of the software should be maintained.
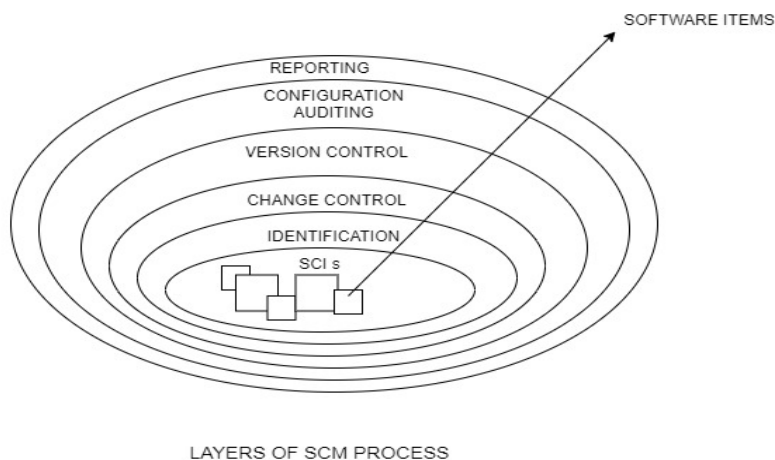
4.6.1.1 Layers Structure of SCM Process

LAYERS OF SCM PROCESS

Inner layer is Software configuration items. Each and every item has specific version. So the first layer is the identification of these items. All the code files, project files, any document which is part of the software development is identified in this layer. After identification of various items, it organizes the items in SCM repository using an object-oriented approach. Objects starts as basic objects and then grouped together into aggregate objects. Each objects has features like name which should be unambiguous to all other objects, a description that contains the SCI types, a project identifier and version information, list of resources needed by the objects, the object realization (i.e., the document, the file, the model etc).

The next layer is the change control. It is a procedural activity that confirms quality and consistency as changes are made to an object. A change request is submitted to a configuration control authority named Change Control Board (CCB).The request is evaluated for technical merit, potential side effects, overall impact on other configuration objects and system functions and projected cost in terms of money, time and resources. An engineering order ECO) is issued for each approved change request. It describes the change to be made, the constraints to follow and the criteria for review and audit.

After that we have version control. Version control is a set of procedure and tools that are required for managing the creation and use of multiple occurrences of objects in the SCM repository. Version control capabilities are:

- All relevant configuration objects are store in **SCM repository**.

- A version management capability that stores all version of a configuration object.

- The software engineer collects all relevant configuration objects and constructs a specific version of the software via make facility.

- Issues or bug tracking capability is also available which enables the team to record and track the status of all outstanding issues associated with each configuration object.

The next higher layer is Configuration Audit. It is an SQA activity that helps to ensure that quality is maintained as changes are made. It

complements the formal technical review and is conducted by the SQA group. It addresses the following questions:

- Whether a technical review conducted to assess technical correctness?

- Whether the software process been followed and software engineering standard been properly maintained?

- Whether the changed been highlighted and documented in the SCI? Have the changed author and change data been specified?

- Whether the change, recording it and reporting it been followed for SCIs item?

- Whether all SCIs been properly updated?

A configuration audit ensures that

- The correct SCIs (by version) have been incorporated into a specific build.

- Documentation is maintained or up to date for the version that has been built.

Next layer we have is Status Reporting also called Status accounting. It provides information about each change to the personnel in an organization with a need to know. Answers what happened, who did it, when did it happen and what else will be affected? The configuration status report

- Placed in an on-line database or on a website for software developers and maintainers to read.

- Given to management and practitioners to keep them appraised of important changes to the project SCIs.

STOP TO CONSIDER

SCM repository maintains a change set. It serves as a collection of all changes made to a baseline configuration. Also used to create a specific version of the software and captures all changes to all files in the configuration along with the reason for changes and details of who made the changes and when.

A meeting is organize for impact analysis and is called Impact Analysis meeting. Meeting is to be done whenever a developer modifies in his code or he is trying to fix a bug or trying to add a new feature into the application or trying to remove features that are already present. Project developer, project tester, project manager are the member of this meeting. By this meeting software tester will know what all areas in the application they need to test. This meeting is usually done by developer or tester. If developer makes any changes to the code, he will call tester to inform in what areas he need to test. Tester can also call this meeting whenever they feel that developers have made changes and they really feel that can get impacted just because of developers tried to add or delete new features. Project manager is a domain expert so he needs to be a part of this meeting.

4.6.3 Software Rejuvenation

Preventive maintenance in the context of software systems has an exciting name called Software rejuvenation. In software engineering, software Rejuvenation is an approach to help prevent performance degradation and other associated values related to aging. It deals software reliability/availability in the operational phase. It includes methods of deciding key performance variables to monitor.It have been adopted as a good practice for many systems. It can reduce costs of sudden aging related failures. Also it can be applied at the discretion of the user/administrator or can be automated.

4.7 SUMMING UP

In short Software Maintenance is widely accepted as a part of System development life cycle. All the modification and updation done after the delivery of software product is Software maintenance. There are number of reasons, why modifications are required. Reasons may be Market conditions such as Policies which changes over time, taxation and newly introduced constraints like how to maintain bookkeeping, may trigger need for modification. Modification may be client requirements such as over time customer may ask for new features or function in the software. Other modification like host modification which include any of the hardware or platform of target host changes. Software changes are needed to keep adaptability.

4.8 ANSWERS TO CHECK YOUR PROGRESS

1. Software maintenance is divided into four categories.

2. Maintenance task is triggered by a maintenance request (change request or error report)

3. Software maintenance task are:

   - changing functions and data of existing code

   - adding new functions and data

   - Reengineering tasks (improving the changeability of the code and data)

   - Optimization tasks (improving the performance of the code

4. Impact analysis is performed to determine which source code members, which documents and which test cases are affected by the change?

5. Maintenance auditing is to ensure that the quality of the software product does not regress as a result of the maintenance tasks.

6. By testing the new code against the data of the old code and comparing the new results with the previous results.

4.9 POSSIBLE QUESTIONS

1. How you can define Software Maintenance?
2. What are the four types of software maintenance according to Lientz and Swanson?
3. What distinguishes adaptive maintenance from corrective maintenance?
4. What distinguishes adaptive maintenance from functional enhancement?
5. What is the purpose of perfective maintenance?
6. What does a Software Maintenance Engineer need to know? List out five items
7. What is the main problem seen in Software maintenance?
8. What does software maintenance include?
9. Describe layers structure of software configuration process.
10. What is impact analysis? How it is different from software rejuvenation?

4.10 REFERENCES AND SUGGESTED READING

- Software Maintenance: Concepts and Practice September 2003. Authors: Penny, Grubb, Armstrong, A. Takang
- Software Rejuvenation: Analysis, Module and Applications, Y. Huang, N. Kolettis and N. Fulton, Proc. FTCS-25,1995
- Software Engineering By Jibitesh Mishra, Ashok Mohanty
- Advances in Software Maintenance Management: Technologies and Solutions Mario Gerardo Piattini Velthuis, Macario Polo, Francisco Ruiz
- Fundamentals of Software Engineering, By Rajib Mall

UNIT 5: SOFTWARE MAINTENANCE MODELS

5.1   INTRODUCTION

A software might need to make change and update in due for its best performance. The changes may be requiring for modify or update any existing solution to create a new solution to a problem in the software. This comes under software maintenance part. This unit reports the need of software maintenance. A training process is a core component of the software maintenance phase. A maintainer should understand the existing code carefully and provide the recent solution for the problem. Different software maintenance model along with its advantages and disadvantages are discussed in this unit along with the challenges of software maintenance.

5.2 UNIT OBJECTIVES

After going through this unit, you will be able to know

- About need of software maintenance.

- About the different software maintenance model.

- About challenges of software maintenance.

Space for learners:

5.3 IMPORTANCE OF SOFTWARE MAINTENANCE

Software might need to make change and update in due for its best performance. The reasons for software maintenance are

1. To fix bugs and error of the software and that is called corrective maintenance.

2. To implement the enhancements, require by users.

3. To increase non-functionalities qualities like performance, security, design, usability of the software.

4. To decrease software complexity.

5. To work in a new environment, upgraded hardware for upgraded operating system, new database management system or other software.

The purpose of software maintenance is to perform the following:

- Expanding the customer requirements and base.

- Enhancing software's capabilities, so that it works in a new environment, hardware, and software.

- Omitting obsolete capabilities by employing newer technology.

5.4 PROCESS OF SOFTWARE MAINTENANCE

Software Maintenance phase of Software Development Life Cycle (SDLC), is implemented through a proper software maintenance process, known as Software Maintenance Life Cycle (SMLC). This life cycle consists of seven different phases, each of which are presented below:

1. **Identification Phase:**

   The software modifications requests are identified, collected, and analysed. Based on the requests, the maintenance activities are scheduled and classified. It is done either by using system or by using log file or error message of the software.

2. **Analysis Phase:**

In this phase, collected software modification requests are analysed for feasibility and scope. A plan is prepared to incorporate the changes in the software. The input attribute, initial estimate of resources, project documentation, cost of modification and maintenance is also estimated in this phase

3. **Design Phase:**

The new modules of the software that need to be replaced or modified are designed based on the requirements received from the different sources. The test cases along with the safety and security issues are also developed for the new design. These test cases are created for the validation and verification of the system.

4. **Implementation Phase:**

In this phase, the modification of the new modules is made in the coding level. New features that demand modification are added, and the modified software along with the new modules is installed.

5. **System Testing Phase:**

Here, the regression testing is performed on the modified modules along with the system to ensure that no defect, error or bug is left undetected. The integration testing is also applied to validate that no new faults are introduced in the software as a result of maintenance activity.

6. **Acceptance Testing Phase:**

Acceptance testing is applied on the system after modifications by the user or by the third party specified by the end user. The testing is used to verify thenewly added features of the software are according to the requirements or not.

7. **Delivery Phase:**

After the successful accomplishing of acceptance testing, the new integrated system is delivered to the user along with the new manual and help files.

CHECK YOUR PROGRESS-I

1. What do you mean by software maintenance?
2. Why do you need to maintain software?
3. Software Maintenance is classified into how many categories?
4. What type of software testing is generally used in Software Maintenance?
5. Which regression test selection technique exposes faults caused by modifications?

5.5 SOFTWARE MAINTENANCE MODELS

To overcome internal as well as external problems of the software, Software maintenance models are used to overcome the different error or issues of a software that are generated though external OR internal sources of a software.

These models use techniques to simplify the process of maintenance as well as to make are cost effective.

1. **Quick-Fix Model:**

   Quick-Fix software maintenance model is an ad hoc model. This is used to identify the issues of software and trying to solve the issues as soon as possible. It performs very quickly and makes the necessary changes in the software to fix the problem as quickly as possible at a low cost. Simple changes of codes are considered in this model to make the impact of changes in the software.
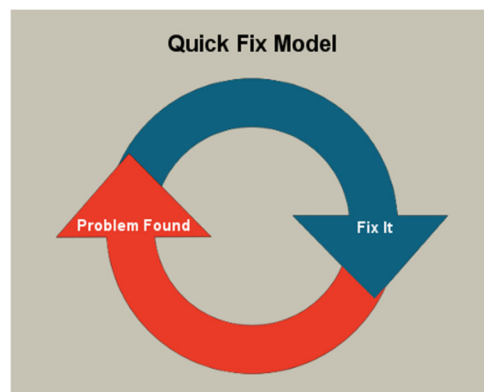
Fig. 5.1 Quick and Fix Model [1]

2. Iterative Enhancement Model:

Iterative enhancement model considers the changes made to software are iterative in nature. The change of the current software depends on the analysis of the existing software system after completing the documents preparation of the existing system at the beginning. Moreover, it attempts to control complexity and tries to maintain good design.Iterative Enhancement Model is divided into three stages:

a. Analysis of existing software system.

b. Classification of required modifications.

c. Implementation of required modifications.
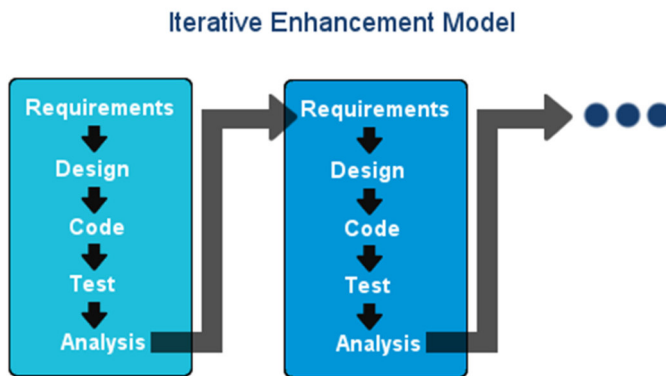
Fig. 5.2 Iterative enhancement Model [1]

3. The Re-use Oriented Model:

In this maintenance model, the necessary part and codes of the existing software system are indentified and reuse for further modification based on the requests. These codes are then going through modification and enhancement for the specified new requirements. The final step of this model is the integration of modified parts into the new system.
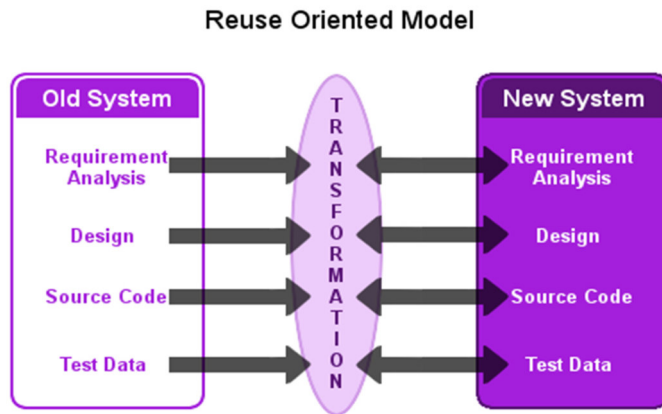
Fig. 5.3 Reuse oriented Model [1]

4. Boehm's Model:

Based on the economic models and principles; the Boehm's Model performs the software maintenance and analysis. A closed loop cycle maintenance steps are used to perform the maintenance of the system wherein the changes are suggested and approved at first. The managements of the software approve the changes, and based on it the necessary changes and maintenance are applied in the system.
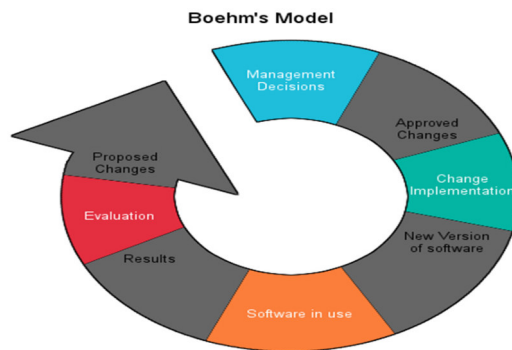
Fig. 5.4 Boehm's Model [1]

5. Taute Maintenance Model:

In 1983, the Taute's model is a maintenance model that developed by Taute which consists of eight phases in cycle fashion. The process of maintenance begins by requesting the change and ends with its operation. It is very easy to understand and undemanding to implement. The phases of Taute's Maintenance Model are:

1. Change request Phase:

   In this phase, the software user makes request in prescribed format to software management team to apply change to software.

2. Estimate Phase:

   The maintenance team or software management team estimatethe time and effort required to apply requested change.

3. Schedule Phase:

   In this phase, management or maintenance team identifies change requests and make schedule for its release and may also prepare documents that are required for planning.

4. Programming Phase:

   The maintenance team ask the programmer to modifies source code of existing software to implement requested change by user and updates all relevant documents like design document, manuals, etc. accordingly.

5. Documentation Phase:

   In this phase, maintenance team ensures the correct changes of the software based on the request received from the user.

6. Release Phase:

   The modified software system along with its documents are delivered to customer.

7. Operation Phase:

   After successful completion of release phase, the software is placed under normal operation and also tried to find the new bugs or issues in the system.A customer may again initiate 'Change request' process in this step.
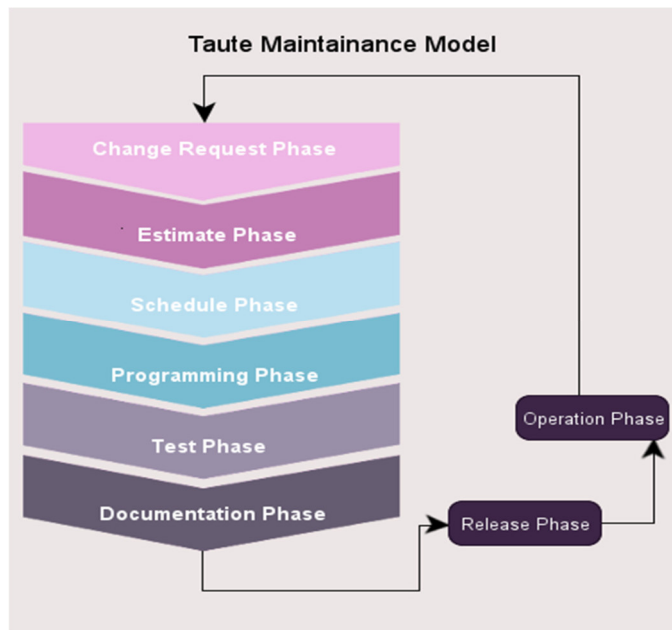
Fig. 5.5Taute's Model [1]

5.6 CHALLENGES OF SOFTWARE MAINTENANCE

The challenges of software maintenance are as follows:

1. **Lack of Traceability**

   o Codes are hardly traceable to the requirements and design specifications.

   o It is very hard for a programmer to identified and correct a serious defect affecting customer operations.

2. **Lack of Code Comments**

   o Codes of a software modules are developed by an individual coder. So, most of the codes are lack of adequate comments. Lesser comments my not help another programmer to understand the codes and its effects.

3. **Obsolete Legacy Systems**

   o The legacy system of a software that provides the backbone of the nation's critical industries, were not designed with maintenance in mind.

o As a consequence, the code supporting these systems is devoid of traceability to the requirements, compliance to design and programming standards.

CHECK YOUR PROGRESS-II

6. How many phases are there in Taute Maintenance Model?
7. The modification of the software to match changes in the ever changing environment, falls under which category of software maintenance?
8. How many types of software maintenance models are found?
9. What is lack traceability of software maintenance?

5.7 SUMMING UP

- Software might need to make change and update in due for its best performance.

- To fix bugs and error of the software and that is called corrective maintenance.

- To increase non-functionalities qualities like performance, security, design, usability of the software, maintenance is required.

- To decrease software complexity, software maintenance is required.

- To work in a new environment, upgraded hardware for upgraded operating system, new database management system or other software, software maintenance is required.

- Software Maintenance phase of Software Development Life Cycle (SDLC), is implemented through a proper software maintenance process, known as Software Maintenance Life Cycle (SMLC).

- SMLC has seven different phases.

- To overcome internal as well as external problems of the software, Software maintenance models are used to overcome the different error or issues of a software that are generated though external OR internal sources of a software.

- Five different software maintenance models are there to perform software maintenance.

- The challenges of software maintenance are as follows

    a. Lack of Traceability

    b. Lack of Code Comments

    c. Obsolete Legacy Systems

5.8 ANSWERS TO CHECK YOUR PROGRESS

1. Software might need to make change and update in due for its best performance. The changes may be requiring for modify or update any existing solution to create a new solution to a problem in the software. This is called as software maintenance.

2. The reasons for software maintenance are

    a. To fix bugs and error of the software and that is called corrective maintenance.

    b. To implement the enhancements, require by users.

    c. To increase non-functionalities qualities like performance, security, design, usability of the software.

    d. To decrease software complexity.

    e. To work in a new environment, upgraded hardware for upgraded operating system, new database management system or other software.

3. Four

4. Regression Testing

5. Inclusiveness

6. Seven

7. Adaptive

8. Five

9. Lack of Traceability means the following

    a. Codes are hardly traceable to the requirements and design specifications.

b. It is very hard for a programmer to identified and correct a serious defect affecting customer operations

5.9 POSSIBLE QUESTIONS

Short answer type questions:

i) What is software maintenance and why does is it important?

ii) What does software maintainer perform?

iii) What are the challenges of software maintenance?

iv) What is the disadvantages of quick and fix software maintenance model?

v) State the difference between identification and analysis phase of software maintenance.

Long answer type questions:

i) Explain the different process of software maintenance.

ii) Explain the different models of software maintenance.

iii) Explain about Taute's software maintenance model.

5.10 REFERENCES AND SUGGESTED READINGS

- https://www.professionalqa.com/software-maintenance-models
- Software Maintenance: Concepts and Practice by Penny Grub

BLOCK III:
SOFTWARE RELIABILITY AND
SOFTWARE MANAGEMENT

UNIT 1: SOFTWARE RELIABILITY

Unit Structure:

1.1 INTRODUCTION

Software reliability is the probability that the software will work without failure for a specified period of time. Failure means the program in its functionality has no met user requirements in some way. Software reliability concerns itself with how well the software functions to meet the requirements of the customer.

-Reliability represents a user-oriented view of software quality. Initially, Software quality was measured by counting the faults in the program and so this approach is developer oriented whereas reliability is user oriented, because, it relates to operation rather than design.

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- To differentiate the failure and faults.

- To highlight the importance of execution and calendar time

- To understand Time interval between failures.

- To understand on the user perception of reliability.

1.3 CONCEPT OF SOFTWARE RELIABILITY

Software reliability is one of the most important elements of the overall quality of any software even after the accomplishment of software work. If it fails to meet its actual performance after its deployment, then the software is considered as unreliable software. We cannot expect better performance from such software.

Software reliability is defined in statistical terms as the probability of failure free operation of a computer program in a specific environment for a specific time for software to be reliable; it must perform operation based upon its analysis and design, available resources, reusability and so on.

If the design standards are not available or the supply is not at proper times, then there is a high probability of software failure or there is degradation in the quality of performance. Therefore, in order to make software reliable, we should take some measures or earlier stages as follows:

- Designing a software project based on available resources.

- Develop software that best fits the current environmental conditions.

- Estimating costs that might be required even after items implementation.

- Estimating the actual performance upon using reusable resources.

The IEEE defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." So it is necessary that software reliability should be measured and evaluated. Though it is hard to achieve software reliability as we don't have a good understanding of the software, it is always tempting to measure something related to reliability to reflect the characteristics. Reliability is a by-product of quality, and software quality can be measured.

1.4 SOFTWARE FAILURE MECHANISMS

Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. There are five different types' software failures. Those are as follows:

- Transient- Transient failures occur only for certain input values while invoking a function of the system.

- Permanent- Permanent failures occur for all input values while invoking a function of the system.

- Recoverable- When recoverable failures occur, the system recovers with or without operator intervention.

- Unrecoverable- In unrecoverable failures, the system may need to be restarted.

- Cosmetic- These classes of failures cause only minor irritations, and do not lead to incorrect results. An example of a cosmetic failure is the case where the mouse button has to be clicked twice instead of once to invoke a given function through the graphical user interface.

While it is tempting to draw an analogy between Software Reliability and Hardware Reliability, software and hardware have basic differences that make them different in failure mechanisms. Hardware faults are mostly physical faults, while software faults are design faults, which are harder to visualize, classify, detect, and

correct. Design faults are closely related to fuzzy human factors and the design process, which we don't have a solid understanding. A partial list of the distinct characteristics of software compared to hardware is listed below:

- **Failure cause**: Software defects are mainly design defects.

- **Wear-out**: Software does not have energy related wear-out phase. Errors can occur without warning.

- **Repairable system concept**: Periodic restarts can help fix software problems.

- **Time dependency and life cycle**: Software reliability is not a function of operational time.

- **Environmental factors**: Do not affect Software reliability, except it might affect program inputs.

- **Reliability prediction**: Software reliability cannot be predicted from any physical basis, since it depends completely on human factors in design.

- **Redundancy**: Can not improve Software reliability if identical software components are used.

- **Interfaces**: Software interfaces are purely conceptual other than visual.

- **Failure rate motivators**: Usually not predictable from analyses of separate statements.

- **Built with standard components**: Well-understood and extensively-tested standard parts will help improve maintainability and reliability. But in software industry, we have not observed this trend. Code reuse has been around for some time, but to a very limited extent. Strictly speaking there are no standard parts for software, except some standardized logic structures.

1.5 SOFTWARE RELIABILITY METRICS

Reliability metrics are used to quantitatively express the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain. The reliability requirements for different categories of software products may be different. For

this reason, it is necessary that the level of reliability required for a software product should be specified in the SRS (software requirements specification) document. In order to be able to do this, some metrics are needed to quantitatively express the reliability of a software product. A good reliability measure should be observer-dependent, so that different people can agree on the degree of reliability a system has. For example, there are precise techniques for measuring performance, which would result in obtaining the same performance value irrespective of who is carrying out the performance measurement. However, in practice, it is very difficult to formulate a precise reliability measurement technique. The next base case is to have measures that correlate with reliability. There are six reliability metrics which can be used to quantify the reliability of software products.

a. **Rate of occurrence of failure (ROCOF)** - ROCOF measures the frequency of occurrence of unexpected behaviour (i.e. failures). ROCOF measure of a software product can be obtained by observing the behaviour of a software product in operation over a specified time interval and then recording the total number of failures occurring during the interval.

b. **Mean Time To Failure (MTTF)** - MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures. Let the failures occur at the time instants t 1, t 2, t n. Then, MTTF can be calculated as

$$\sum_{i=1}^{n} \frac{t_{i+1-}t_i}{(n-1)}$$

It is important to note that only run time is considered in the time measurements, i.e. the time for which the system is down to fix the error, the boot time, etc. are not taken into account in the time measurements and the clock is stopped at these times.

c. **Mean Time To Repair (MTTR)** - Once failure occurs, sometime is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.

d. **Mean Time Between Failure (MTBR)** - MTTF and MTTR can be combined to get the MTBR metric: MTBF = MTTF + MTTR. Thus, MTBF of 300 hours indicates that once a failure occurs,

the next failure is expected after 300 hours. In this case, time measurements are real time and not the execution time as in MTTF.

e. **Probability of Failure on Demand (POFOD)** - Unlike the other metrics discussed, this metric does not explicitly involve time measurements. POFOD measures the likelihood of the system failing when a service request is made. For example, a POFOD of 0.001 would mean that 1 out of every 1000 service requests would result in a failure.

f. **Availability**- Availability of a system is a measure of how likely shall the system is available for use over a given period of time. This metric not only considers the number of failures occurring during a time interval, but also takes into account the repair time (down time) of a system when a failure occurs. This metric is important for systems such as telecommunication systems, and operating systems, which are supposed to be never down and where repair and restart time, are significant and loss of service during that time is important.

1.6 SOFTWARE RELIABILITY MEASUREMENT TECHNIQUES

Reliability metrics are used to quantitatively express the reliability of the software product. The option of which parameter is to be used depends upon the type of system to which it applies & the requirement of the application domain. Measuring software reliability is a severe problem because we don't have a good understanding of the nature of software. It is difficult to find a suitable method to measure software reliability and most of the aspects connected to software reliability. Even the software estimates have no uniform definition. If we cannot measure the reliability directly, something can be measured that reflects the features related to reliability. Software reliability techniques can be divided into four categories:

1.6.1 Product Metrics

Product metrics are those which are used to build the artefacts, i.e. requirement specification documents, system design documents, etc.

These metrics help in the assessment if the product is right sufficient through records on attributes like usability, reliability, maintainability & portability. In these measurements are taken from the actual body of the source code.

1.6.2 Project Management Metrics

Project management metrics define project characteristics and execution. If there is proper management of the project by the programmer, then this helps us to achieve better products. A relationship exists between the development process and the ability to complete projects on time and within the desired quality objectives. Cost increase when developers use inadequate methods. Higher reliability can be achieved by using a better development process, risk management process, configuration management process. These metrics are:

- Number of software developers.

- Staffing pattern over the life-cycle of the software

- Cost and schedule

- Productivity

1.6.3 Process Metrics

Process metrics quantify useful attributes of the software development process & its environment. They tell if the process is functioning optimally as they report on characteristics like cycle time & rework time. The goal of process metric is to do the right job on the first time through the process. The quality of the product is a direct function of the process. So process metrics can be used to estimate, monitor and improve the reliability and quality of software. Process metrics describe the effectiveness and quality of the processes that produce the software product. Examples: The effort required in the process

- Time to produce the product

- Effectiveness of defect removal during development

- Number of defects found during testing

- Maturity of the process

1.6.4 Fault and Failure Metrics

A fault is a defect in a program which appears when the programmer makes an error and cause failure when executed under particular conditions. These metrics are used to determine the failure-free execution software.

To achieve this objective, a number of faults found during testing and the failures or other problems which are reported by the used after delivery are collected, summarized and analysed.

Failure metrics are based upon customer information regarding faults found after release of the software. The failure data collected is therefor used to calculate failure density, Mean Time between Failures (MTBF), or other parameters to measure or predict software reliability.

1.7 SOFTWARE RELIABILTY IMPROVEMENT TECHNIQUES

Good engineering methods can largely improve software reliability. In real situations, it is not possible to eliminate all the bugs in the software; however, by applying sound software engineering principles software reliability can be improved to a great extent. Three approaches are used to improve reliability of the software. These approaches are:

i.   **Fault avoidance:** The design and implementation phase of the software development uses the process that minimizes the probability of faults before the software is delivered to the user.

ii. **Fault detection and removal:** Verification and validation techniques are used to detect and remove faults. In addition, testing and debugging can also remove faults.

iii. **Fault tolerance:** The designed software manages faults in such a way that software failure does not occur. There are three aspects of fault tolerance. These are :

a.   **Damage assessment :** This detects parts of software affected due to the occurrence of faults

b. **Fault recovery:** This restores the software to the last known safe state. Safe state can be defined as the state where the software functions as desired

c. **Fault repair:** This involves modifying the software in such a way that faults does not occur.

1.8 SOFTWARE FAULT TOLERENCE

Software fault tolerance is the ability for software to detect and recover from a fault that is happening or has already happened in either the software or hardware in the system in which the software is running to provide service by the specification. Software fault tolerance is a necessary component to construct the next generation of highly available and reliable computing systems from embedded systems to data warehouse systems. To adequately understand software fault tolerance, it is important to understand the nature of the problem that software fault tolerance is supposed to solve.

Software faults are all design faults. Software manufacturing, the reproduction of software, is considered to be perfect. The source of the problem being solely designed faults is very different than almost any other system in which fault tolerance is the desired property

1.9 SOFTWARE FAULT TOLERENCE TECHNIQUES

Software fault-tolerance techniques are used to make the software reliable in the condition of fault occurrence and failure. There are three techniques used in software fault-tolerance. First two techniques are common and are basically an adaptation of hardware fault-tolerance techniques.

1.9.1 N-version Programming

In this technique, n versions of a program are developed by n developers. All these copies are run simultaneously, and the one with the most fault tolerance is selected. This is a fault-detection technique used at the developing stage of the software. In N-version programming; N versions of software are developed by N

individuals or groups of developers. N-version programming is just like TMR in hardware fault-tolerance technique. In N-version programming, all the redundant copies are run concurrently and result obtained is different from each processing. The idea of n-version programming is basically to get the all errors during development only.

1.9.2 Recovery Blocks

This technique is somewhat the same as above, except for the redundant copies are not run simultaneously. They are run one by one and are generated with a different set of algorithms. This technique is used where task deadlines are more than the computation time. Recovery blocks technique is also kike the n-version programming but in recovery blocks technique, redundant copies are generated using different algorithms only. In recovery block, all the redundant copies are not run concurrently and these copies are run one by one. Recovery block technique can only be used where the task deadlines are more than task computation time.

1.9.3 Check-pointing and Rollback Recovery

This technique is different from above two techniques of software fault-tolerance. In this technique, system is tested each time when we perform some computation. This technique is basically useful when there is processor failure or data corruption.

1.10 SOFTWARE RELIABILTY MODELS

A software reliability model indicates the form of a random process that defines the behavior of software failures to time.

Software reliability models have appeared as people try to understand the features of how and why software fails, and attempt to quantify software reliability. Over 200 models have been established since the early 1970s, but how to quantify software reliability remains mostly unsolved. There is no individual model that can be used in all situations. No model is complete or even representative.

Most software models contain the following parts:

- Assumptions

- Factors

A reliability growth model is a numerical model of software reliability, which predicts how software reliability should improve over time as errors are discovered and repaired. Although there are different reliability models, three simple ones are discussed in this section.

1.10.1 Jelinski-Moranda Model

This model is credited with being the first reliability model. It belongs to a class of exponential order statistic model that assumes that fault detection and correction begins when a program contains N faults and all the faults have the same rate φ. The basic assumptions of the model are:

1. The program contains N initial faults which are an unknown but fixed constant.

2. Each fault in the program is independent and equally likely to cause a failure during a test.

3. Time intervals between occurrences of failure are independent of each other.

4. Whenever a failure occurs, a corresponding fault is removed with certainty.

5. The fault that causes a failure is assumed to be instantaneously removed, and no new faults are inserted during the removal of the detected fault.

6. The software failure rate during a failure interval is constant and is proportional to the number of faults remaining in the program.

The program failure rate at the ith failure interval is given by,

$$\lambda(t_i) = \phi[N - (i - 1)], \quad i = 1, 2, ..., N$$

Where

ϕ = a proportional constant, the contribution any one fault makes to the overall program

N = the number of initial faults in the program

t_i = the time between the $(i-1)^{th}$ and the $(i)^{th}$ failures.

For example, the initial failure intensity is

$$\lambda(t_1) = \phi N$$

and after the first failure, the failure intensity decreases to

$$\lambda(t_2) = \phi(N-1)$$

and so on.

The partial distribution function (pdf) of t_i is

$$f(t_i) = \phi[N-(i-1)]e^{-\phi(N-(i-1))t_i}$$

The cumulative distribution function (cdf) of t_i is

$$F(t_i) = 1 - e^{-\phi[N-(i-1)]t_i}$$

The software reliability function is, therefore,

$$R(t_i) = e^{-\phi(N-i+1)t_i}$$

1.10.2 Musa's Basic Execution Time Model

This model was established by J.D. Musa in 1979, and it is based on execution time. The basic execution model is the most popular and generally used reliability growth model, mainly because:

- It is practical, simple, and easy to understand.
- Its parameters clearly relate to the physical world.
- It can be used for accurate reliability prediction.

The basic execution model determines failure behavior initially using execution time. Execution time may later be converted in calendar time.

- The failure behavior is a nonhomogeneous Poisson process, which means the associated probability distribution is a Poisson process whose characteristics vary in time

- It is equivalent to the M-O logarithmic Poisson execution time model, with different mean value function

- The mean value function, in this case, is based on an exponential distribution.

Variables involved in the Basic Execution Model:

- Failure intensity (λ): number of failures per time unit.

- Execution time (τ): time since the program is running.

- Mean failures experienced (μ): mean failures experienced in a time interval.

In the basic execution model, the mean failures experienced μ is expressed in terms of the execution time (τ) as

$$\mu(\tau) = v_0 \; x \left(1 - e^{-\frac{\lambda_0}{v_0} \tau} \right)$$

where,

-λ_0: stands for the initial failure intensity at the start of the execution.

-v_0: stands for the total number of failures occurring over an infinite time period; it corresponds to the expected number of failures to be observed eventually.

The failure intensity expressed as a function of the execution time is given by

$$\lambda(\tau) = \lambda_0 x \, e^{-\frac{\lambda_0}{v_0} \tau}$$

It is based on the above formula. The failure intensity λ is expressed in terms of μ as:

$$\lambda(\tau) = \lambda(\mu) = \lambda_0 \left(1 - \frac{\mu}{v_0} \right)$$

where,

λ_0: Initial

v_0: Number of failures experienced, if a program is executed for an infinite time period.

μ: Average or expected number of failures experienced at a given period of time.

τ: Execution time.

The Musa basic execution time model assumes that all faults are equally likely to occur, are independent of each other and are actually observed. The execution times between failures are modelled as piecewise exponentially distributed. The intensity function is proportional to the number of faults remaining in the program and the fault correction rate is proportional to the failure occurrence rate.

1.10.3 Goel-Okumoto (GO) Model

The model developed by Goel and Okumoto in 1979 is based on the following assumptions:

1. The number of failures experienced by time t follows a Poisson distribution with the mean value function μ (t). This mean value method has the boundary conditions **μ(0) = 0 and Lim$_{t \to \infty}$μ(t) = N < ∞.**

2. The number of software failures that occur in **(t, t+Δt]** with **Δt → 0** is proportional to the expected number of undetected errors, **N - μ(t)**. The constant of proportionality is ∅.

3. For any finite collection of times **t$_1$ < t$_2$ < · · · < t$_n$** the number of failures occurring in each of the disjoint intervals **(0, t$_1$),(t$_1$, t$_2$)... (t$_{n-1}$,t$_n$)** is independent.

4. Whenever a failure has occurred, the fault that caused it is removed instantaneously and without introducing any new fault into the software.

Since each fault is perfectly repaired after it has caused a failure, the number of inherent faults in the software at the starting of testing is equal to the number of failures that will have appeared after an infinite amount of testing. According to assumption 1, M (∞) follows a Poisson distribution with expected value N. Therefore, N is the expected number of initial software faults as compared to the fixed but unknown actual number of initial software faults μ$_0$ in the Jelinski Moranda model.

Check Your Progress-1

1. Write the characteristics of software failures over hardware failures?

2. What are the objectives of software reliability models?

3. Write the difference between Goel-Okumoto model and Jelinski Moranda Model?

4. Why MUSA's model is most popular among reliability models?

State TRUE or FALSE:

1. A failure can be related to the operating system.

2. Defect that causes error in operation or negative impact is called fault.

3. Programmers can make mistakes while developing the source code.

4. A model used to describe software reliability is known as MTBF

5. Availability is the probability of software to opearate and deliver the desired request.

1.11 SUMMARY

- Software reliability is the probability that the software will work without failure for a specified period of time.

- Software failure is classified into Transient, Permanent, Recoverable, Unrecoverable and Cosmetic

- Reliability metrics are used to quantitatively express the reliability of the software product.

- ROCOF measures the frequency of occurrence of unexpected behavior.

- MTTF is the average time between two successive failures, observed over a large number of failures.

- MTTR measures the average time it takes to track the errors causing the failure and to fix them.

- POFOD measures the likelihood of the system failing when a service request is made.

- Availability of a system is a measure of how likely shall the system is available for use over a given period of time.

- Software reliability techniques can be divided into four categories, namely product metrics, project management metrics, Process metrics, fault and failure metrics.

- Fault tolerance techniques mainly N-Version Programming, Recovery blocks and Check-pointing and Rollback Recovery

1.12 ANSWERS TO CHECK YOUR PROGRESS

1.13 POSSIBLE QUESTIONS

Short answer type questions:

1. Define Software Reliability?

2. What are the types of software failure?

3. Differentiate between software reliability and hardware reliability?

4. Which factor is MUSA's basic model based on?

5. What are the software the fault tolerance techniques?

Long answer type questions:

1. What is the need of reliability matrices? Explain the different reliability matrices?

2. Explain the software reliability measurement techniques?

3. Explain the software reliability improvement techniques?

4. Write a note on Jelinski-Moranda Model and Goel-Okumoto Model

5. Describe MUSA's basic execution time model?

1.14 REFERENCES AND SUGGESTED READINGS

- Musa, J.D, A. Iannino and K.okumoto Software Reliability : Measurement, Prediction , Application, professional Edition : Software Engineering Series , McGraw- Hill, NewYork , NY.,1990

- Software Reliability by John Musa.

- Software Metrics and Reliability by Linda Rosenburg, Ted Hammer, Jack Shaw
  satc.gsfc.nasa.gov/support/ISSRE_NOV98/
  software_metrics_and_reliability.html – 2

- Software Reliability by Jiantao,
  http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

UNIT 2: SOFTWARE QUALITY MANAGEMENT

Unit Structure:

2.1 INTRODUCTION

If the quality of the software does not meet the standards, then the final product might be slow, inconsistent, not reusable and hence, difficult to perform the maintenance. In manufacturing industry, the concept of 'quality assurance' and 'quality control' were already in use, and later these concepts were incorporated in software industry as well. Quality assurance (QA) defines the processes and standards that lead to high-quality products whereas quality control is the application of these processes to eliminate the products that do not meet the required level of quality. Thus quality management works in tandem with the software development process as shown in fig 7.1. The quality management process performs periodical checks so that the project meets the required standard. The quality management team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.

Software Development Process

D1   D2   D3   D4   D5

Quality Management Process

Standard and Procedures   Quality Plan   Quality Review Reports
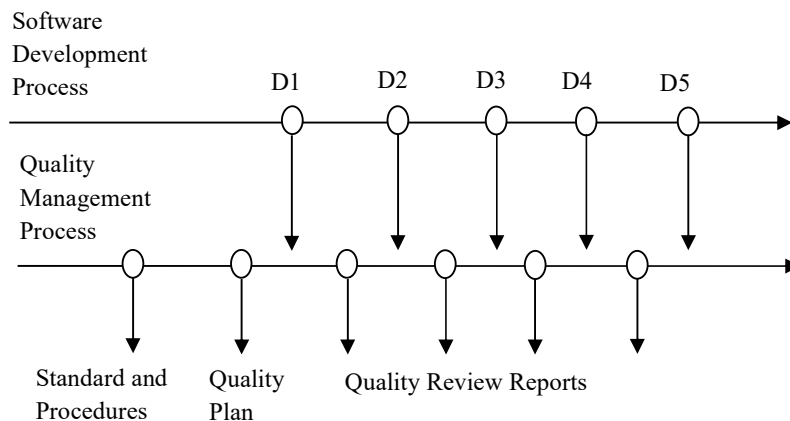
Fig. 7.1: Quality management process

2.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand what is software quality management and why is it necessary.

- Comprehend the importance of standards in the software quality management and know how standards are used in quality assurance.

- Understand how reviews and inspections improve software quality.

- Learn how quality is maintained in Agile software development process.

2.3 SOFTWARE QUALITY

In manufacturing industry, the product is manufacture under a set of standard processes and procedures; and once the product is manufactured, it is tested to check whether it meets the standard required specifications. If there are products that miss the standard specifications by a little margin, then such products will still be deemed alright because there is a high probability that customers might never notice or even ignore slight niggles or objections. But quality of software cannot be evaluated the way quality of manufactured products are determined because one, the software

developer and the customer may never converge in their view of the requirements. Second, it might never be possible to satisfy the requirements of all the thousands of users and stakeholders. And finally, in some cases, it might not be possible to directly assess certain quality parameters like maintenance. Thus, the assessment of software quality is a subjective process where it is the responsibility of the quality management team to use their judgement to decide if an acceptable level of quality has been achieved and whether the software can be used for its intended purpose. To do this we have to answer the following questions about the system's characteristics.

a) Have programming and documentation standards been followed in the development process?

b) Has the software been properly tested?

c) Is the software sufficiently dependable to be put into use?

d) Is the performance of the software acceptable for normal use?

e) Is the software usable?

f) Is the software well-structured and understandable?

Thus, the quality management team analyses the tests that has been carried out, makes sure the test are correctly done and examines the test results to check whether the desired goal has been acquired. But, software quality is not just about whether the system requirement has been successfully acquired, but also whether the non-functional system attributes has been evaluated upon. There are around 15 important software quality attributes, as shown in table 7.1, on which we have to test our system.

Sl. no.	Attribute	Significance
1	Complexity	Software complexity describes a specific set of characteristics that focus on how our piece of code or software interacts with other pieces of code.
2	Efficiency	Software Efficiency determines the amount of code and testing resources required by a program to perform a particular function.
3	Interoperability	It is the ability of two or more systems to

		communicate or exchange data easily and to use the data that has been exchanged.
4	Learnability	Learnability is a quality of software products and interfaces that allows users to quickly become familiar with them and able to make good use of all their features and capabilities
5	Modularity	Modularity is the degree to which a system's components may be separated and recombined easily.
6	Portability	It is the ability of a software application to run on numerous platforms such as data portability, hosting, viewing, etc.
7	Reliability	It relates to the software's ability to continue operating, under given environmental conditions, for a particular amount of time.
8	Resilience	Software resilience refers to the ability to absorb the impact of disruption in one or more parts of a system, while still continuing to provide an acceptable level of service.
9	Reusability	Reusability is the likelihood that a segment of source code can be used again to add new functionalities with slight or no modification.
10	Robustness	Robustness is the ability of the system to handle with abnormalities like errors during execution, erroneous input, sudden changes in hardware or software environment, etc.
11	Scalability	It is the ability of a system to handle the demand for stress caused by increased usage without decreasing performance.
12	Security	It is the ability of a system to resist or block malicious or unauthorized attempts that destroy the system and at the same time provide access to legitimate users.
13	Supportability	It is the ability of a system that satisfies necessary requirements and needs to identifying and solving problems.
14	Testability	It shows how well the system or component facilitates to perform tests to determine whether the predefined test criteria have been met.

15	Usability	It is described as how the user is utilizing a system effectively and the ease of which users can learn to operate or control the system. One of the well-known principles of usability is to make it very user-friendly and keep it as simple as possible.

Table 7.1: Software quality attributes

Particular software may not be able to stand strong on all the above mentioned attributes — for example, if we want to improve robustness then this might lead to loss of efficiency. Therefore, the quality management team should try to find out what are the major criteria for the system being developed. Say, for example, if efficiency is very important then there is reason why other criteria like scalability or usability may be relaxed. This should be stated in the quality plan so that the engineers working on the software development can cooperate to achieve this. Thus, the quality plan should include all the definition of the quality assessment process and the agreed upon quality attributes that are essential for the software.

STOP TO CONSIDER

1. The quality of software cannot be evaluated the way quality of manufactured products are determined.

2. The quality management process goes in tandem with the software development process.

3. The assessment of software quality is a subjective process.

4. Software quality is not just about whether the system requirement has been successfully acquired, but also depends on many non-functional system attributes.

5. The quality management team should try to find out what are the essential non-functional attributes for the software.

CHECK YOUR PROGRESS-1

1. Why the quality management team should be independent from the development team?

2. The assessment of software quality is a subjective / objective process. (Choose the correct option)

3. Why should the quality management team try to find out the major software attributes that are significant for the system being developed?

2.4 SOFTWARE STANDARDS

Software standards play a major role in software quality management. As assessment of software quality is a subjective process and it depends on the expectations of the organization and the user; so, having a set of specified standards will assist in setting the definition of 'quality' for the particular software. It will be the guiding principles for all the engineers in an organization and will also assist a new engineer to quickly adapt, in case anyone leaves the organization.

In software engineering, there are two types of standards:

a) Product standard: These are the standards that the product being developed should have. It includes document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used

b) Process standard: These are the standard processes that should be followed during software development. It includes good development practice, definitions of specification, design and validation processes, process support tools, and a description of the documents that should be written during these processes.

Many international bodies like the Institute of Electrical and Electronics Engineers (IEEE), American National Standards Institute (ANSI) and British Standards Institution (BSI) have developed software standards that can be used for our projects. There are standards even for programming languages such as Java and C++. There are standards that encompass many software

engineering terminologies like procedures for deriving and writing software requirements, quality assurance procedures, software verification and validation processes (IEEE, 2003) and safety / security-critical procedures as IEC 61508 (IEC,1998).

When developing the standards for a company, the quality management team should formulate the standards by obtaining information and insight from such well-established international quality standards. This way they can set the standards that will be credible and justifiable. But all software standards might not be applicable to all kinds of software projects. The quality management team should plan and prepare in advance, what are the criteria and standards that are viable for the individual project.

They should be able to identify what are the standards that are absolutely necessary, what can be used but in a modified manner and what can be completely rejected. If this is not done in advance, then the software engineers will be wasting their time in achieving the standard that is not necessary rather than fulfilling the required goals and standards for the project.

STOP TO CONSIDER

1. Software standards will act as the guiding principles for all the engineers in an organization.
2. In software engineering, there are two types of standards: product standard and process standard.
3. The quality management team should formulate the standards by obtaining information and insight from well-established international quality standards.

CHECK YOUR PROGRESS-2

4. Software standard will be the _____ for all the engineers in an organization.

5. What are some of the international standard bodies?

6. How can we set credible software standard?

2.5 REVIEWS AND INSPECTIONS

Reviews and inspections is another means by which we can verify the quality of software. First the review team validates whether the quality standards have been followed or not. The review team should look into the documents pertaining to software specifications, designs, or code, process models, test plans, configuration management procedures, process standards, and user manuals. They should check the consistency and completeness of the documents and the programming code and makes sure that quality standards have been followed. After they verify the conformity to standards, then they try to discover problems and omissions in the software or project documentation. That is, they thoroughly examine the records and documentations of the procedures used to discover errors and omissions.

2.5.1 Review Process

The review process mainly consists of three phases:

a) *Pre-review activities*: In pre-review activities there is planning and preparation of the review. A review team is created, a time and place for the review is set, and the documents to be reviewed are distributed. Individual review team members will go through the relevant documents and work independently to find errors, omissions, and deviations from standards.

b) *Review meeting*: A review meeting is formally arranged. It is chaired by one of the review team members while another one transcribes all the important review decisions and actions to be taken. One of the authors of the document will have to explain all the details about the document and the project. The chair should ensure that all the comments, instructions and future actions are duly recorded.

c) *Post-review activities*: After the review meeting is over, it is time to address the issues and problems raised during the review. For example there may be issues related to solving bugs in software, or restructuring the code so that it becomes more readable, more efficient, less complex and easier to maintain. Sometimes there may be issue related to compliance of quality standards and rewriting the document, or in some cases allocating more resources to the project

if needed. The review chair must ensure that all the review comments have been duly processed and thoroughly examined.

2.5.2 Program Inspections

The main aim of program inspection is to find bugs in the program being developed. A completed program is not required to perform the inspection; even an incomplete version of the system can be examined. In fact, the most effective way to inspect a system is to use its test cases, find problems and thus improve its effectiveness. One of the main tasks of the members of the inspection team is to meticulously review each line of the source code, find any logical errors or any anomalies and show these at an inspection meeting.

Several books and journals have been published which provide us a list of common programming errors that can be found in a particular project or domain. In fact, different list exists for different programming language as each programming language can have different types of errors or anomalies. One such book is "*Managing the Software Process*", by Watts S. Humphrey, Addison-Wesley publication (1989). The inspection team members can make use of such "list of common errors" to examine the project under preview.

STOP TO CONSIDER

1. The review team verifies whether the software conforms to quality standards and also tries to discover problems, anomalies and omissions in the software.
2. The review process mainly consists of three phases.
3. The main aim of program inspection is to find bugs in the program under inspection.
4. The inspection team can make use of already existing list of common errors and problems.

2.6 QUALITY MANAGEMENT AND AGILE DEVELOPMENT

Although, most companies agree that inspections are very effective in finding bugs, but some may forsake this process. Software engineers with experience of program testing may disagree that inspections can be more effective than testing. Again, sometimes

managers may not have the liberty to sustain extra cost; as inspections may incur additional costs during design and development. Another such example is 'Agile' processes which does not approve of using formal inspection methods.

The Agile method focuses more on code development than on formal inspection documentation process. It relies more on individual programmer's ethical and responsible frame of mind. Here each member follows good coding practises like refactoring, avoiding deep nesting, limiting line length, using proper naming conventions, etc, as well as having a test-driven development so that a high quality code is created. In the Agile processes, team members can also cooperate among themselves to check each other's code. They follow the rules:

a) *Check before check-in*: It suggests that programmers should always check their own code, consult with other team members, before using it in a system.

b) *Never break the build*: It suggests that a programmer's code should never break the build. So, they have to rigorously test their code before they use it. If the build is broken, then it is the programmer's responsibility to fix it.

c) *Fix problem when you see them*: If a programmer detects a problem or anomaly in another programmer's code, then he or she can directly modify the code instead of resending the code to the original programmer.

STOP TO CONSIDER

1. The formal inspection process may not be adopted by all companies.
2. In the Agile processes, team members cooperate among themselves to check each other's code.
3. In the Agile processes, each programmer should rigorously test their own code, consult with other team members, before the code is checked in to build the system.

CHECK YOUR PROGRESS-3

7. The review team makes sure that _____ have been followed and checks whether there are _____ in the software.

8. For program inspection, a completed program is required to perform the inspection. (True or False).

9. One of the main tasks of the members of the inspection team is to meticulously review each line of the source code. (True or False).

10. The Agile method focuses more on _____ than on _____.

11. Agile method relies heavily on individual programmer's ethical and responsible frame of mind to write good documentation. (True or False).

2.7 SUMMING UP

- Software development process and software quality management process should be done simultaneously.

- Software quality cannot be evaluated the way quality of manufactured products are determined.

- The assessment of software quality is a subjective process depends on lot of other non-functional factors like Complexity, Efficiency, Interoperability, Learnability, Modularity, Portability, Reliability, Resilience, Reusability, Robustness, Scalability, Security, Supportability, Testability and Usability.

- One of the aims of the quality management team is to find out those the essential non-functional attributes which directly affects the software.

- Setting standards will assist in defining the 'quality' for the software, and help the engineers to follow the established guidelines.

- Insight and information from well-established international quality standards will guide the quality management team to formulate the standards for the company.

- The purpose of review process is to verify whether the software conforms to quality standards and also to discover problems, anomalies and omissions in the software.

- The review process mainly consists of three phases: Pre-review activities, Review meeting and Post-review activities.

- The main aim of program inspection is to find bugs in the program by meticulously examining each line of source code.

- The inspection team can make use of already existing list of common errors and anomalies.

- The Agile software development method focuses more on code development than on formal inspection documentation process and relies heavily on individual programmer's ethical and responsible behaviour.

- 12. The programmers in Agile software development process follow the rules like *Check before check-in*, *Never break the build* and *Fix problem when you see them*.

2.8 ANSWERS TO CHECK YOUR PROGRESS

1. The quality management team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.

2. The assessment of software quality is a <u>subjective</u> process.

3. A software may not be able to satisfy all the non-functional attributes. So, the quality management team, in consultation with all the stakeholders, should try to find out what are the attributes that are more essential for the system. This should be stated in the quality plan, so that the software engineers can cooperate to achieve this.

4. Software standard will be the <u>guiding principles</u> for all the engineers in an organization

5. Some of the international standard bodies are Electrical and Electronics Engineers (IEEE), American National Standards Institute (ANSI) and British Standards Institution (BSI).

6. When developing the standards for a company, the quality management team should formulate the standards by obtaining

information and insight from such well-established international quality standards. This way they can set the standards that will be credible and justifiable.

7. The review team makes sure that <u>quality standards</u> have been followed and checks whether there are <u>problems and omissions</u> in the software.

8. False.

9. True

10. The Agile method focuses more on <u>code development</u> than on <u>formal inspection documentation process</u>.

11. False

2.9 POSSIBLE QUESTIONS

Short answer type questions:

1. State why the evaluation process for the quality of software cannot be measured in the same way as the quality of manufactured products.

2. To decide if an acceptable level of quality has been achieved what are some of the questions about the system's characteristics that the quality management team needs to ask?

3. Why do you think the assessment of software quality is a subjective process?

4. What are two types of standards in software engineering?

5. Why should the quality management team formulate the standards from well-established international quality standards?

6. Explain the three phases of review process.

7. State the rules followed in Agile software development.

Long answer type questions:

1. Explain some of the non-functional system attributes that the software quality needs to be evaluated upon.

2. What is the importance of software quality standards and how to achieve it?

3. What is the significance of reviews and inspections and how can you implement them?

4. Explain how software quality is maintained in Agile method of software engineering.

2.10 REFERENCES AND SUGGESTED READINGS

- Wixom, B. H., Roth, R. M., (2008), Systems Analysis and Design, Wiley Publishing

- Kendall, K. E., Kendall J. E., (2019), Systems Analysis and Design, Pearson

Space for learners:

UNIT 3: SOFTWARE CONFIGURATION MANAGEMENT

Unit Structure:

3.1 INTRODUCTION

As the software is being developed there might be scenarios where there is sudden change in system requirements and some additional features might have to be incorporated in the existing system. Occurrence of new bugs or arrival of a new version of hardware may also force the developer to adapt their software according to the current changes. As changes are inevitable, each of versions software has to be maintained and managed properly. If it is not maintained properly then it will lead to a chaotic situation where no one will have the correct idea about what changes has been made and which version is supposed to be considered. Thus configuration management plays a vital role in software development. There are four components of software configuration management namely, *Change management*, *Version management*, *System building*, and *Release management*; all of these will be discussed in this current unit.

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand that sudden change in system requirements will continuously emerge through the software development process.

- Learn how to analyse, respond and manage changes in the system.

- Learn to perform version management using codeline and baseline.

- Understand the complex process of system building.

- Understand the differences between a system version and a system release, and learn about release management process.

3.3 CHANGE MANAGEMENT

As changes to a system due to detection of new bugs, arrival of newer hardware or need for additional system requirements is almost certain to emerge, so it is utmost necessary to make sure that the changes are applied in a controlled and managed process. The main aim here is to find out which changes are commendable to make, which are vital, sensible as well as cost-effective and more importantly keeping track of the components that has been changed.

Any stakeholder be it developer, user, product owner or project manager can request or report for a change in a system. This can be straightforward bug report, or request for additional system requirements. The request has to be formally reported through a change request form (CRF). After the CRF has been submitted, it is evaluated and determined whether the change request is notable. In some events it may happen that a bug or an additional requirement has been reported, but by the time the report has been filed, the bug might have been already erased or the additional feature has already been implemented but the requester may not know about it. Sometimes the additional feature may already be existing in the system and due to some misunderstanding, they may not be aware of it. In such cases the request is dissolved and it not processed any further.

If the requested change is a valid claim then it has to be duly processed and the following steps are taken. First, vital information have to be collected like date of submission of the report, the significance of each change requested, what will be the impact and scope of the change, which component of the system will be affected, what will be the cost incurred and finally whether the request is approved or rejected. If it is approved then it has to be decided how the changes have to be implemented and where the changes have to be incorporated. For example, the change may have to be included in any one or more of the following cases: requirements documentation, technical design documentation, programming code, project schedules, test cases, etc. As the change to the respective component is being made, it is very essential that a snapshot of the changed state is recorded and properly maintained. This is called as derivation history. For example if a change is made in program source code, then a good way of keeping the derivation history is by using standardized comment at the begin of the code. This comment will point to the change request that caused the change. For documents, records of changes incorporated in each version are usually maintained in a separate page at the front of the document.

STOP TO CONSIDER

1. Any stakeholder can request or report for a change in a system.
2. Changes to an existing system are inevitable, so each of versions software has to be maintained and managed properly.
3. The request has to be formally reported through a change request form (CRF).
4. Valid requests are appropriately processed and analysed whereas misunderstood or superfluous requests are rejected.
5. For every changes made, derivation history is to be maintained.

CHECK YOUR PROGRESS-1

1. What are the sudden changes that might occur during the software development process?

2. What are the four components of software configuration management?

3. What are the main objectives of software configuration management?

4. Any stakeholder be it developer, user, product owner or project manager cannot request or report for a change in a system. (True or False).

3.4 VERSION MANAGEMENT

The main objective of version management (VM) is to keep track of different versions of software components and the systems in which these components are used. It should be ensured that no two developers have conflict or interference with the changes made to their respective versions. In other words, version management is the process of managing codelines and baselines.

3.4.1 Codelines and Baselines

When a programmer revises or changes the source code of a component in a system, then there will be different versions of each component. A **codeline** is the sequence of versions of source code where the later version in the sequence is derived from the earlier version. Before checking in, i.e., submitting, the modified code, a programmer has to follow certain guidelines. There has to be a limit to the number of times a programmer can check in the code. A programmer cannot be allowed to check in as many times as possible because it will create a huge confusion for the whole team. Before the code is checked in or submitted, it should be tested as much as possible and integrated as early as possible.

Once a stable version of the components in a system are judiciously saved by all the programmers or developers, then a **baseline**, or to put in in simple terms, a milestone or a snapshot of the system, is created and stored. Here we have to ensure that all the information like who has made the version and what changes were made in the

version has to well-documented and astutely preserved. At no point of time should the versions in a baseline be lost, so that if a previous version of the software is requested it can be effortlessly returned. Apart from component versions, the baseline also includes a specification of the libraries used, configuration files, etc. Fig. 8.1 depicts a diagrammatical view of versions, codelines and baselines.
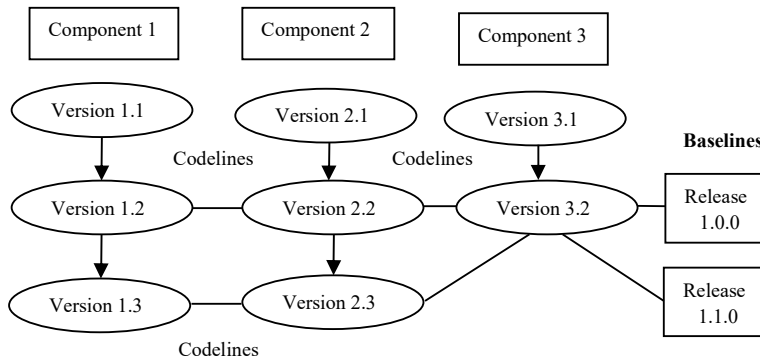
Fig 8.1: Versions, Codelines and Baselines

3.4.2 Version Control System

There are software tools that enable us to identify, store and control access to the different versions of components. These are called as Version Control (VC) systems and there are two main categories of version control system: *Centralized* and *Distributed*. In centralized system there is a single master repository that maintains all versions of the software components that are being developed. In distributed systems multiple versions of the component repository can be stored at the same time. Each user has its own repository and working copy. Example of centralized VC system is *Subversion*, whereas example of distributed VC system is *Git*.

Key features of version control systems:

- Version and release identification
- Change history recording
- Support for independent development
- Project support
- Storage management

STOP TO CONSIDER

1. The main objective of version management (VM) is to keep track of different versions of software components and the systems in which these components are used.

2. A codeline is the sequence of versions of source code where the later version in the sequence is derived from the earlier version.

3. A baseline represents the stable and agreed upon versions of all components in a certain point of time.

4. There are software tools called Version Control (VC) systems that enable us to identify, store and control access to the different versions of components.

CHECK YOUR PROGRESS-2

5. _____ ensures that no two developers have conflict or interference with the changes made to their respective versions.

6. A _____ is the sequence of versions of source code whereas a

_____ is a stable and agreed upon versions of all components in a certain point of time.

7. A programmer need not follow any guidelines to check in his version of code. (True or False).

8. In a baseline, a previous version of the software cannot be requested. (True or False).

9. Apart from component versions, the baseline also includes a specification of the libraries used, configuration files, etc. (True or False).

10. Give an example of centralized version control system and distributed version control system. (Subversion, Git)

3.5 SYSTEM BUILDING

System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc. As the version management system contains the repositories of component versions, there need to exist communication between system building tools and version management tools. The system building tool also needs configuration description of baselines existing in the system.

System building has three main platforms:

a) Development system: It includes development tools such as compilers, source code editors, etc. Here, the developers need to create a private workspace where they can download a copy of the code (also called as check out) from the code repository located in the version management system. Once they have made the changes in their code and before they again commit it back into the version management system, they have to test it in their private development environment. For this they will have to use the local build tools in their private workspace.

b) Build server: It is used to build definitive, executable versions of the system. Once the developers have made the changes, performed appropriate testing, they upload their code (also called as check in) to the version management system. The system build may rely on external libraries that are not included in the version management system.

c) Target environment: It is the actual platform on which the software actually has to run. The actual platform can be the same type of hardware and software environment used during the development and build phase. But many times the target environment can be rather smaller and simpler system like a mobile phone than a high-end system used in the development environment.

3. 5. 1 Automated Build Tools

The build script is a definition of the system to be built. It includes information about components and their dependencies, and the versions of tools used to compile and link the system. The build script includes the configuration specification which ensures that the scripting language and the configuration description language remain the same. The configuration language includes constructs to describe the system components to be included in the build and their dependencies.

System building involves assembling a large amount of information about the software and its operating environment. Therefore, for large systems, automated build tool can be used to create a system build. Some example of automated build tools are Bazel, Jenkins,

Apache Maven, Gradle, Gulp, Travis CI, Nant, etc. Some of the features that the automated build tools may provide are as follows:

a) Build script generation: It analyzes the program that is being built, identifies dependent components, and automatically generates a build script. It may also support the manual creation and editing of build scripts.

b) Version management system integration: It verifies the required versions of components from the version management system.

c) Minimal recompilation: It finds out which part of the source code requires to be recompiled and performs the compilation.

d) Executable system creation: It links the compiled object code files with each other and with other required files, such as libraries and configuration files, and creates an executable system.

e) Test automation: Some build systems can automatically run automated tests using test automation tools such as JUnit. These check that the build has not been 'broken' by changes.

f) Reporting: It provides reports about the success or failure of the build and the tests that have been run.

g) Documentation generation: It may also be able to generate release notes about the build and system help pages.

STOP TO CONSIDER

1. System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc.

2. System building has three main platforms: development system, build server and target environment.

3. The build script is a definition of the system to be built which includes information about components and their dependencies, and the versions of tools used to compile and link the system.

4. For larger complex systems, automated build tool can be used to create a system build.

5. Automated build tools provide many features like build script generation, version management system integration, minimal recompilation, executable system creation, test automation, reporting and documentation generation.

CHECK YOUR PROGRESS-3

11. To build a complete system, there should be communication between _____ tools and _____ tools.

12. System building consists of development platform where the developer creates a private workspace and checks in the code to the version management system. (True or False).

13. How does a developer use the development platform in system building?

14. Once appropriate testing has been performed, the developer has to check in the code to the version management system and build an <u>executable version</u> of the system.

15. The actual platform is always the same as the one used during the development and build phase. (True or False).

16. What is a build script?

17. Name some automated build tools.

3.6 RELEASE MANAGEMENT

A system release is a version of a software system that is distributed to customers or users. Two kinds of release types are there: major release and minor release. Major release contains significant new functionality whereas minor release is used to find bugs or customer / user related issues and repair them. Minor releases are usually distributed free of charge while the user has to make payment to use major releases.

A software company may have various releases of a product and may distribute these releases to many customers or users. Sometimes a customer may be comfortable in using an older release and after many years of use, may want specific changes to be made in that particular release. So it becomes imperative for the software company to manage all the different releases, maintain information about which release is provided to which customer, store information between releases and system versions and should be able to correlate and regulate the software that has been delivered to a particular customer.

To document a release, the specific versions of the source code components that were used to create the executable code have to be properly recorded. Copies of the source code files, corresponding executables, and all data and configuration files should be properly stored. Even the versions of the operating system, libraries, compilers, and other tools used to build the software should be well documented because these may be required to build exactly the same system at some later date. So along with the source code, copies of the platform software and the tools used to create the system should be stored in the version management system.

Release distribution is a costly affair as it requires advertising and publicity materials, marketing strategies, awareness campaigns, etc., in order to entice customers into buying the new release of the software. If the releases are too frequent then the customer may not bother to frequently buy new releases, or if the new releases with improved features are seldom made then it will be difficult to hold on to the customers as they may start using latest software from another company.

While making a new release, a company cannot blindly believe that all the customers will be using their latest release. In fact, there may be many customers who may be just satisfied with the older release and may never bother to upgrade to the latest release. Say for example, a company has three releases of their software, R.1, R.2 and R.3, with R.1 being the oldest and R.3 being the newest release. Say, R.1 is simple basic release, R.2 requires the customer to upload some specialized data and R.3 is an advanced distributed system which makes use of the data uploaded by the customer in R.2. Now, since release R.3 requires data uploaded by the customer in R.2 then, if a customer wants to go directly from R1 to R3, then the company has to maintain proper record about which release was provided to which customer and accordingly take the correct set of actions. If this is not handled appropriately, then the software may crash, causing frustration for the customer and embarrassment for the company. To avoid such scenarios, some companies may even try to implement automatic update of their software whenever there is any new release available. But if the customer switches off the automatic update feature then this also cannot be forcefully implemented.

STOP TO CONSIDER

1. Two kinds of release types are there: major release and minor release.

2. It is important for the company to manage all the different releases and maintain the information about which release is provided to which customer.

3. Along with the source code, copies of the platform software and the tools used to create the system should be stored in the version management system.

4. Release distribution is financially costly.

5. A company cannot presume that all the customers will be using their latest release.

CHECK YOUR PROGRESS-4

18. Major release contains significant new <u>functionality</u> whereas minor release is used to find new <u>bugs</u>.

19. Major releases are usually distributed free of charge while the user has to make payment to use minor releases. (True or False).

20. A company should store information between releases and system versions and should be able to correlate and regulate the software that has been delivered to a particular customer. (True or False).

21. Why a company should not make too frequent or too less releases?

22. A company can always trust the customer to have their latest release installed. (True or False).

3.7 SUMMING UP

- As the software is being developed there might be scenarios where there is sudden change in system requirements, new bugs may be detected or new hardware may have to be introduced.

- So, changes to a system are inevitable and hence each of versions software has to be maintained and managed properly

otherwise it will lead to a chaotic situation where no one will have the correct idea about what changes has been made and which version is supposed to be considered.

- There are four components of software configuration management namely, Change management, Version management, System building, and Release management.

- The main aim of change management is to find out which changes are commendable to make, which are vital, sensible as well as cost-effective and more importantly to keep track of the components that has been changed.

- Any stakeholder can request or report for a change in a system by formally filling a change request form (CRF).

- Valid requests are appropriately processed and analysed whereas misunderstood or superfluous requests are rejected.

- The main objective of version management (VM) is to keep track of different versions of software components and the systems in which these components are used.

- Managing codelines and baselines is an important activity of version management process.

- Version Control Software tools like Subversion and Git can be used to identify, store and control access to the different versions of components.

- System building is the complicated process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc.

- System building has three main platforms: development system, build server and target environment.

- For larger projects automated build tools can be used, which provide many features like build script generation, version management system integration, minimal recompilation, executable system creation, test automation, reporting and documentation generation.

- A system release is a version of a software system that is distributed to customers or users. It is of two types: major release and minor release.

- Major release contains significant new functionality whereas minor release is used to find bugs or customer / user related issues and to repair them.

- It is very important for the software company to manage all the different releases, maintain information about which release is provided to which customer, store information between releases and system versions.

- Release distribution incurs a lot of costs in order to advertise the new releases.

- A company cannot presume that all the customers will always be using their latest release.

3.8 ANSWERS TO CHECK YOUR PROGRESS

1. Some of the sudden changes that might occur during the software development process are additional features to be included, changes in system requirements, occurrences of new bugs or arrival of new hardware, etc.

2. The four components of software configuration management are change management, version management, system building, and release management.

3. The objectives of software configuration management are to find out which changes are commendable to make, which are vital, sensible as well as cost-effective and more importantly keeping track of the components that has been changed.

4. False

5. Version management ensures that no two developers have conflict or interference with the changes made to their respective versions.

6. A codeline is the sequence of versions of source code whereas a baseline is a stable and agreed upon versions of all components in a certain point of time.

7. False.

8. False.

9. True.

10. An example of centralized version control system is *Subversion* and an example of distributed version control system is *Git*.

11. To build a complete system, there need to exist communication between <u>system building</u> tools and <u>version management</u> tools.

12. False.

13. The developers need to create a private workspace where they have to check out their code from the code repository located in the version management system. Once they have made the changes in their code and before they can again commit it back into the version management system, they have to test it in their private development environment. For this they will have to use the local build tools in their private workspace.

14. Once appropriate testing has been performed, the developer has to check in the code to the version management system and build an <u>executable version</u> of the system.

15. False.

16. The build script is a definition of the system to be built which includes information about components and their dependencies, and the versions of tools used to compile and link the system. The build script includes the configuration specification which ensures that the scripting language used is the same as the configuration description language.

17. Some example of automated build tools are Bazel, Jenkins, Apache Maven, Gradle, Gulp, Travis CI, Nant, etc.

18. Major release contains significant new <u>functionality</u> whereas minor release is used to find new <u>bugs</u>.

19. False

20. True

21. If the releases are too frequent then the customer may not bother to frequently buy new releases, or if the new releases with improved features are seldom made then it will be difficult to hold on to the customers as they may start using latest software from another company.

22. False.

3.9 POSSIBLE QUESTIONS

Short answer type questions:

1. Why is it utmost necessary to properly maintain and manage the changes applied to each version of software?

2. Why is change request form (CRF) used for?

3. Explain briefly how misunderstood or superfluous requests are handled in software configuration management?

4. Explain succinctly what is the goal of version management?

5. What are Centralized and Distributed version control system?

6. What is system building and why is it a complicated process?

7. State the difference between major and minor release.

8. Why is it important for the company to manage all the different releases and maintain the information about which release is provided to which customer?

9. How will you document and record a release?

Long answer type questions:

1. Explain in detail how to skilfully supervise *change management* in software configuration management.

2. Explain how you will perform version management using codeline and baseline.

3. Explain in detail what is the significance of Version Control System?

4. Explain in detail, the three platforms used by developers in system building.

5. Describe the features of automated build tools that will assist developers in system building process.

6. Explain the importance of release management and why company cannot presume that all the customers will be using their latest release?

7. Explain all the process and procedures involved in release management.

3.10 REFERENCES AND SUGGESTED READINGS

- Pressman, R. S., (2004), Software Engineering, A Practitioner's Approach, McGraw-Hill

- Kendall, K. E., Kendall J. E., (2019), Systems Analysis and Design, Pearson

Space for learners: