UG CA S6-02 DSC

FOUR YEAR BACHELOR'S DEGREE COURSE
(CBCS in Accordance with NEP 2020)

Semester VI

SOFTWARE ENGINEERING

COMPUTER SCIENCE

Hiranya Chandra Bhuyan School of Science and Technology
Krishna kanta Handiqui State Open University

Subject Experts

Prof. Shikhar Kumar Sarma, Department of Information Technology, Gauhati University
Prof. Hiren Kumar Deva Sarma, Department of Information Technology, Gauhati University
Dr. Sanjib Kumar Kalita, Department of Computer Science, Gauhati University

Programme Coordinator : Dr. Tapashi Kashyap Das, KKHSOU

Course Coordinators:

Dr. Tapashi Kashyap Das, KKHSOU
Dr. Sruti Sruba Bharali, KKHSOU

Contributors

UNITS	Author
1, 14	Dr. Kshirod Sarmah, PDUAM, Amjonga
2	Dr. Tapashi Kashyap Das, KKHSOU
3	Mr. Mridul Suklabaidya, National Informatics Centre
4	Dr. Dwipen Laskar, Gauhati University
5	Dr. Atowar Ul Islam, USTM
6, 12	Dr. Sangeeta Kakoty, KKHSOU
7, 8	Dr. Antara Malakar, Assam Engineering College
9	Dr. Samarjit Das, Cotton University
10	Dr. Brajen Kumar Deka, Assam Don Bosco University
11	Dr. Sonia Sarmah, Assam Don Bosco University
13	Dr. Tarali Kalita, NERIM Group of Institutions

Editorial Team

Content editing: Dr. Meenakshi Gogoi, Girijananda Chowdhury University(Unit 1-9,12,14-15)
Dr. Sruti Sruba Bharali, KKHSOU (Unit 10, 11, 13)

Language editing: Prof. (Retd.) Robin Goswami, Cotton College

Format editing: Dr. Sruti Sruba Bharali, KKHSOU

Print: May, 2026 ***ISBN:***

© Krishna Kanta Handiqui State Open University

 This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

Headquarters: Patgaon, Rani Gate, Guwahati-781017
City Office: NH 37, Resham Nagar, Khanapara, Guwahati 781022;
Web: www.kkhsou.in

Table of Contents

Unit		Page
Unit 1:	**Introduction to Software Engineering** Software Engineering, Need of Software Engineering, Software Characteristics, Software Development Projects: Programs versus Products, Quality and Productivity of software products.	7-15
Unit 2:	**System Development Methodologies** System Development Phases, Software Life Cycle Model, Software Development Models: Classical and Iterative Waterfall Model, Prototyping Model, Evolutionary Model, Spiral Model, Comparison of different Models.	16-25
Unit 3:	**Software Project Management** Software Project Manager, Project Planning, Project Estimation Techniques, Empirical method of Estimation, Heuristic method of Estimation- COCOMO Model, Analytical method of Estimation, Risk Management, Configuration Management.	26-46
Unit 4:	**Staffing and Scheduling** Staffing, Scheduling, Work Breakdown Structure, Activity Networks, Critical Path, Gantt Charts, PERT Charts	47-58
Unit 5:	**Role of System Analyst** System Analyst, Knowledge and Qualities of System Analyst, Role of a System Analyst.	59-69
Unit 6:	**Requirement Analysis and Specification** Fact Gathering, Requirement Analysis, Feasibility Study and Types, User Transaction Requirement, User Design Requirements, System Requirement Specification (SRS), Functional Requirements of SRS, Organization of SRS.	70-81
Unit 7:	**Software Design** Modules Concepts and Types of Modules, Structured Chart, Qualities of Good Design: Coupling, Types of Coupling, Cohesion, Types of Cohesion, Approaches to Software Design, Function-Oriented Design, Object-Oriented Design	82-91
Unit 8:	**Function-Oriented Software Design** Entity-Relationship Diagrams, Decision Tree and Decision Table, Data Flow Diagrams (DFD), Data Dictionary: Elements of DD, Advantage of DD; Pseudo code, Input And Output Design.	92-106

Unit 9: **Object-Oriented Software Design** **107-117**
Unified Modeling Language (UML), UML Diagrams, Use Case Model, Class Diagrams, Activity Diagrams.

Unit 10: **User Interface Design** **118-135**
Characteristics of a Good User Interface, Basic Concept: User Guidance and Online Help, Modeless Interface, Graphical User Interface (GUI) vs. Text-based User Interface, Types of User Interfaces, Fundamentals of Component-based GUI Development, User Interface Design Methodology.

Unit 11: **Software Coding** **136-148**
Coding, Coding Standards and Guidelines, Code Review: Code Walkthrough, Code inspection, Clean room testing, Software Documentation, Internal Documentation, External Documentation.

Unit 12: **Software Testing** **149-157**
Software Testing, Testing Strategies, Types of Testing: Black-Box Testing, White-Box Testing, Stress Testing, Storage Testing, Performance Testing, Unit Testing, Integration Testing.

Unit 13: **Software Reliability and Quality Management** **158-172**
Software Reliability, Hardware versus Software Reliability, Reliability Metrics, Reliability Growth Modelling, Statistical Testing, Software Quality, Software Quality Management System, Evolution of Quality Systems, Product Metrics versus Process Metrics

Unit 14: **Software Maintenance** **173-179**
Software Maintenance, Types of Software Maintenance, Software Reverse Engineering, Maintenance Cost

Unit 15: **Case Tools** **180-186**
Introduction to CASE, Building Blocks of CASE, Integrated CASE Environment.

Course Introduction: Software Engineering

Software has become an integral part of modern society, powering everything from business operations and communication systems to healthcare, education, transportation, and entertainment. This course on "**Software Engineering"** introduces learners to the fundamental concepts, principles, methodologies, and practices involved in developing high-quality software systems. It provides a comprehensive understanding of the software development process, from problem identification and requirement analysis to design, implementation, testing, deployment, and maintenance.

The course covers various software development life cycle models, project planning and management techniques, staffing and scheduling methods, and the critical role of system analysts in software projects. Learners will gain knowledge of requirement engineering, software design methodologies, user interface design, coding standards, and documentation practices. The course also emphasizes function-oriented and object-oriented design approaches, including the use of modeling tools such as Data Flow Diagrams (DFDs), Entity-Relationship Diagrams (ERDs), and Unified Modeling Language (UML) diagrams for effective software design and communication. In addition, the course explores software testing strategies, reliability assessment, quality management systems, maintenance activities, and Computer-Aided Software Engineering (CASE) tools that support software development activities.

The Self Learning Material (SLM) is written in a simple and learner-friendly manner. Each unit includes explanations, examples, illustrations, exercises, and self-assessment questions to help learners understand the concepts easily and study independently. By the end of the course, learners will be equipped with the knowledge and skills required to participate effectively in software development projects, apply established engineering principles to software creation, and contribute to the development of reliable, maintainable, and high-quality software products in professional environments.

UNIT 1 : INTRODUCTION TO SOFTWARE ENGINEERING

UNIT STRUCTURE

1.1 Learning Objectives

1.2 Introduction

1.3 Software Engineering

1.4 Need of Software Engineering

1.5 Software Characteristics

1.6 Software Programs vs. Software Product

1.7 Quality and Productivity of Software Products

1.8 Let Us Sum Up

1.9 Further Readings

1.10 Answers to Check Your Progress

1.11 Model Questions

1.1 LEARNING OBJECTIVES

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

- define software engineering
- describe the importance of software engineering
- define software characteristics
- differentiate between software programs and software products

1.2 INTRODUCTION

In this unit we will introduce the concept of software engineering. Software engineering is concerned with theories, methods and tools for professional cost-effective software development. In this unit, we will learn about the needs for and characteristics of software engineering. In the next unit, we will explore the concepts related to software development methodologies.

1.3 SOFTWARE ENGINEERING

We need to know a liitle bit about software before we learn about software engineering. Software is more than a program code; it is considered to be a collection of executable

program codes associated with essential libraries and documentations. A software, when it is made for a special and specific requirement is known as **Software Product**. On the otherhand, **ENgineering** is all about developing products using well defined scientific principles and methods.While developing a system or software product, it is important to go through a series of engineering steps that help the developer to create a high-quality product on time. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are essentials to achieve a good quality software.

The process of developing a software product using software engineering principles and techniques are known as **Software Evolution** as shown in figure 1.1.



Figure 1.1: Software Evolution Process

Software Evolution includes the initial development of software and its maintainance and updates till the desired software product is developed, which satisfies the expected the requirements.

There are many definitions of software engineering which are given by several authors and institutions. Some important definitions are given below:

- According to ***IEEE Comprehensive Definition,*** software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software.
- According to ***Barry Boehm,*** software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedure and associated documentation.
- According to ***Fritz Bauer,*** a German computer scientist, software e ngineering is the establishment and use of sound engineering principles in order to obtain

economically the software products which are reliable and which work efficiently on real machines.

Some other definitions are as follows :

- Software Engineering is an engineering discipline which is concerned with all aspects of software production.
- Software Engineering is a discipline whose aim is the production of fault free software that satisfies the user's needs and that is delivered on time and within budget.
- Software Engineering is concerned with the theories, methods and tools that are needed to develop the software products in a cost effective way.

1.4 NEED OF SOFTWARE ENGINEERING

Software engineering is an engineering approach for software development process. We can alternatively view it as a systematic collection of past experiences. The experiences are arranged in the form of methodologies and guidelines. Without using software engineering principles it would be difficult to develop large applications. In industry, it is usually needed to develop large programs to accommodate multiple functions. A problem with developing such large software applications is that the complexity and difficulty levels of the software increase exponentially with their sizes. In such situations software engineering techniques come to rescue. Software engineering helps to reduce the programming complexity. Software engineering principles use two important techniques to reduce problem complexity: ***abstraction*** and ***decomposition***.

The principle of *abstraction* implies that a problem can be simplified by omitting irrelevant details. In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose. Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction and so on. Abstraction is a powerful way of reducing the complexity of the problem. The other approach to tackle problem complexity is *decomposition*. In this technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one. However, in this technique any random decomposition of a problem into smaller parts will not help. The problem has to be decomposed in such a way that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution. A good decomposition of a problem should minimize interactions among various components. If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.

The needs of software engineering in some other essential facts are given below:

- For the large software product developments.
- For scalability of the software product developments.
- For low cost high value software product.
- For maintaining dynamic and quality management software product.

1.5 SOFTWARE CHARACTERISTICS

A software product can be considered as good when it offers good reliable service to the user. Some important fundamental characteristics of a good software product must satisfy the following three factors:

(i) Operational (ii) Transitional and (iii) Maintenance

Operational tells how the software works in operation which can be measured on the basis of budget,usability, efficiency, correctness, functionality, flexibility, security and safety.

Transitional tells how efficiently the software moves from one plateform to another that can be measured by portability, interoperability, reusability and adaptability.

Maintenance tells how efficiently the software has its capabilities to maintain itself based on modularity, maintainability,flexibilty and scalability.

To gain an understanding of software and ultimately an understanding of software engineering, it is important to examine the characteristics of software that makes it different from other things that human beings have built. When hardware is built, the human creative process (analysis, design, construction, testing) is ultimately translated into a physical form. Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different from those of hardware.

- ***Software is developed or engineered, it is not manufactured in the classical sense.***

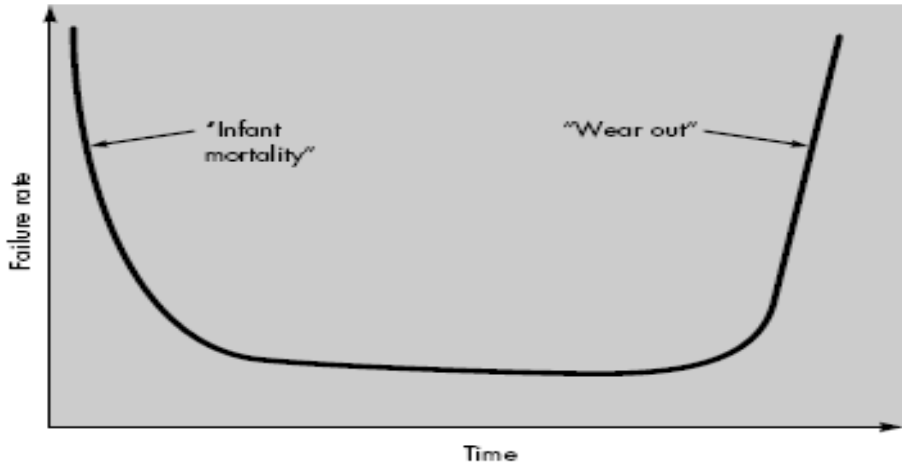


Figure 1.2: Failure rate curve for software

Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software. Both activities are dependent on people, but the relationship between the people applied and the work accomplished is entirely different. Both activities require the construction of a "product" but the approaches are different. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

- **Software doesn't "wear out."**

Figure 1.2 depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental disorders. Stated simply, the hardware begins to wear out the software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity than hardware maintenance.

Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the "idealized curve" shown in figure 1.3. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear—software does not wear out. But it does deteriorate. This seeming contradiction can best be explained by considering the "actual curve" shown in figure 1.3.

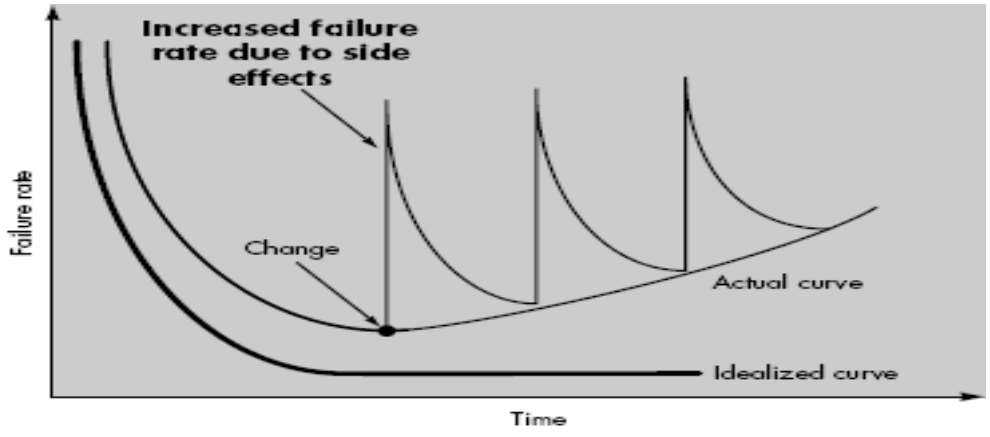


Figure 1.3: Failure rate curve for software

During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in figure 1.3. Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change. Another aspect of wear illustrates the difference between hardware and software. When a hardware component wears out, it is replaced by a spare part.

1.6 SOFTWARE PROGRAMS vs. PRODUCT

Programs are developed by individuals for their personal use. They are therefore, small in size and have limited functionality but software products are extremely large. In case of a program, the programmer himself is the sole user but on the other hand, in case of a software product, most users are not involved with the development. In case of a program, a single developer is involved but in case of a software product, a large number of developers are involved. For a program, the user interface may not be very important, because the programmer is the sole user. On the other hand, for a software product, user interface must be carefully designed and implemented because developers of that product and users of that product are totally different. In case of a program, very little documentation is expected, but a software product must be well documented. A program can be developed according to the programmer's individual style of development, but a software product must be developed using the accepted software engineering principles. The difference between program and software product has been depiced in figure 1.4 and figure 1.5 respectively.

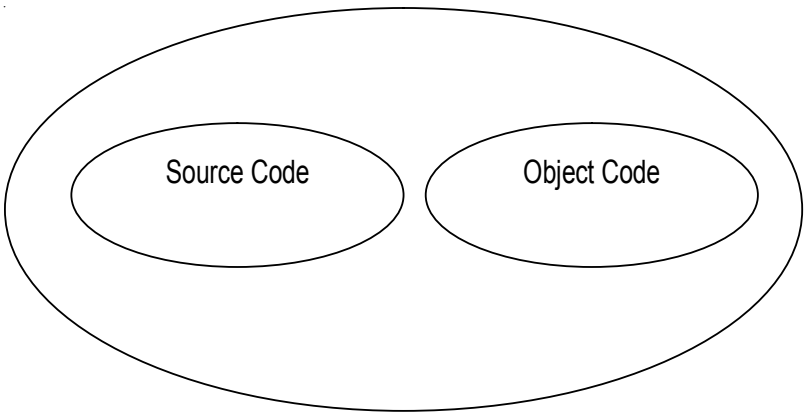


Figure 1.4: Program = Source Code + Object Code

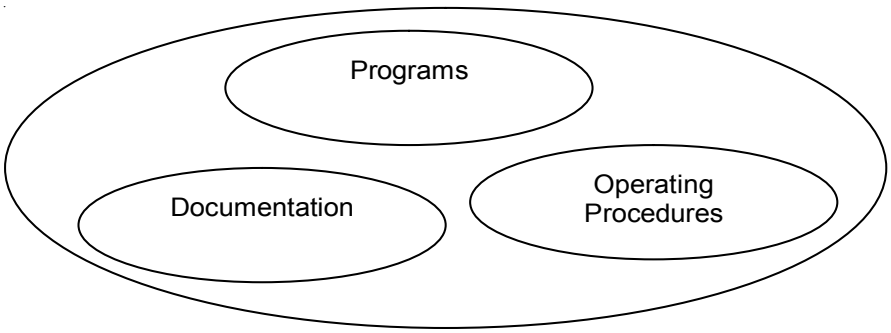


Figure 1.5: Software = Program + Documentation + Operating Procedures

1.7 QUALITY AND PRODUCTIVITY OF SOFTWARE PRODUCTS

An engineering discipline, almost by definition, is driven by practical parameters of cost, schedule, and quality. A solution that takes enormous resources and many years may not be acceptable. Similarly, a poor-quality solution, even at a low cost, may not be of much use. Like all engineering disciplines, software engineering is driven by the three major factors: **cost, schedule, and quality.**

The **cost** of developing a system is the cost of the resources used for the system, which, in the case of software, is dominated by the manpower cost, as development is largely labor-intensive. Hence, the cost of a software project is often measured in terms of person-months, i.e., the cost is considered to be the total number of person-months spent in the project. (Person-months can be converted into a dollar amount by multiplying it with the average dollar cost, including the cost of overheads like hardware and tools, of one person-month).

Schedule is an important factor in many projects. Business trends are dictating that the time to market a product should be reduced; that is, the cycle time from concept to delivery should be small. For software, this means that it needs to be developed faster.

The other major factor driving any production discipline is **quality.** Today,
quality is a main mantra, and business strategies are designed around quality. Clearly, developing high-quality software is another fundamental goal of software engineering. However, while cost is generally well understood, the concept of quality in the context of software needs further discussion.

CHECK YOUR PROGRESS

Q1. Fill in the blanks :

a) Two important techniques used in software engineering for reducing problem

complexity are _____ and _____.

b) Software is not _____ to the environmental maladies.

c) Software engineering is driven by three major factors _____, _____ and _____.

d) A complite software program is consists of source code and _____.

e) Software = Programme + _____ + Operating Procedures.

1.8 LET US SUM UP

- Software engineering is an engineering approach for software development. Without using software engineering principles it would be difficult to develop large programs.
- Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.
- In case of a program, a single developer is involved but in case of a software product, a large number of developers are involved. For a program, the user interface may not be very important, because the programmer is the sole user. On the other hand, for a software product, user interface must be carefully designed and implemented because developers of that product and users of that product are totally different.
- Software engineering is driven by the three major factors: *cost, schedule*, and *quality*.
- Some important fundamental characteristics of a good software product must satisfy the following three factors:
  (i) Operational (ii) Transitional and (iii) Maintenance
- Software engineering principles use two important techniques to reduce problem complexity: abstraction and decomposition.

1.9 FURTHER READINGS

- Jawdekar, W. S. (2004). Software Engineering Principles and Practices.
- Leach, R. J. (2016). Introduction to software engineering. CRC Press.
- Jalote, P. (2012). An integrated approach to software engineering. Springer Science & Business Media.

✓ 1.10 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: a) abstraction, decomposition

b) susceptible

c) cost, schedule and quality

d) object code

e) documentation

? 1.11 MODEL QUESTIONS

Q1. Briefly software engineering.

Q2. What is the need of software engineering?

Q3. What are the characteristics of software? Describe briefly.

Q4. Distinguish between a program and a software product.

Q5. Describe briefly the factors on which quality and productivity of a software product depends.

****

UNIT 2 : SYSTEM DEVELOPMENT METHODOLOGIES

UNIT STRUCTURE

2.1 Learning Objectives
2.2 Introduction
2.3 System Development Phases
  2.3.1 Requirement Analysis
  2.3.2 Feasibility Study
  2.3.3 System Design
  2.3.4 Development
  2.3.5 Implementation
  2.3.6 Testing and Maintenance
2.4 Need for a Software Life Cycle Model
2.5 System Development Approaches
  2.5.1 Waterfall Model
  2.5.2 Prototyping Model
  2.5.3 Spiral Model
2.6 Let Us Sum Up
2.9 Further Readings
2.8 Answers to Check Your Progress
2.9 Model Questions

2.1 LEARNING OBJECTIVES

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

- learn about system development as a life cycle approach
- describe the various stages of system/software development
- learn several models of the system development

2.2 INTRODUCTION

In the previous unit, we have learnt about topics like software engineering, software characteristics and quality. In this unit, we will learn about system development life cycle and the various stages associated with the life cycle. There are various system development approaches, also called *System Development Life Cycle Model*. Each model follows a particular life cycle in order to ensure success in the process of system development. In this unit, we will present three such approaches used in software

development. In the next unit, we will explore the issues related to software project management.

2.3 SYSTEM DEVELOPMENT PHASES

System development is regarded as another form of problem solving in software engineering which consists of activities like understanding the problem, deciding a plan for a solution, coding and testing the coded program. For a successful development of a system, the total development process is devided into smaller activities called *phase* or *stage*.

A system development life cycle is a sequence of different activities that take place during development. The first stage in the life cycle of any system is usually the ***requirement analysis*** stage. Generally, the subsequent stages are : ***feasibility study, system design***, ***development***, ***implementation***, ***testing*** and ***maintenance***. We are now going to give a brief description of the above mentioned phases.

2.3.1 Requirement Analysis

All possible requirements of the system to be developed are captured in this phase. Requirements are set of functionalities and constraints that the end-user (who will be using the system) expects from the system. The requirements are gathered from the user of the system. These requirements are analyzed for their validity and for the possibility of incorporating the requirements in the system to be developes. Finally, a system requirement specification (SRS) document is prepared which serves the purpose of guideline for the next phase of the model.

2.3.2 Feasibility Study

The *feasibility study* is an important step in any system development process. The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product. At first, system analysts or project managers try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the system. After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution. Based on this analysis they pick the best solution and determine whether the solution is

feasible financially and technically.

2.3.3 System Design

Before starting the actual coding, it is highly important to understand what we are going to create and what it should look like? The requirement specifications from first phase are studied in this phase and system design is prepared. System design should stress on the following three activities:

- Data design,
- Process design, and
- User interface design

The *data design* activity focuses on the design of the logical structure of database and files to be used by the proposed information system. *Process design* activity focuses on the design of the software resources, that is, the programs and procedures needed by the proposed information system. An *interface design* activity focuses on designing the interactions betwen end users and computer systems; System Design helps in specifying the hardware and system requirements and also helps in defining the overall system architecture. The system design specifications serve as input for the next phase of the model.

2.3.4 Development

On receiving system design documents, the work is divided into modules/units and actual coding is started. The system is first developed in small programs called modules, which are then integrated. Code generation can be accomplished at this stage. Programming tools like *compilers, interpreters, debuggers* etc... are used to generate the code. Different high level programming languages like *C, C++, Pascal, Java* are used for coding. With respect to the type of application, the right programming language is chosen. As a result, the system is developed.

2.3.5 Implementation

The system implementation stage involves hardware and software acquisition, site preparation, user training and installation of the system. It must be realised that implementation may be the most crucial phase of system development life cycle, because this step is vital in assuring the success of any newly developed system. Even a well-designed system will fail if it is not properly implemented.

2.3.6 Testing and Maintenance

Once the code is generated and implemented, the software program testing begins.

Different testing methodologies are available to unravel the bugs that were committed during the previous phases. Some companies build their own testing tools. After successfully testing the software, it is delivered to the customer.

Generally, problems with the system developed arise after its practical implementation. So the issues related to the system are solved after the deployment of the system. The software will definitely undergo change once it is delivered to the customer. Not all the problems come in picture directly but they arise from time to time and needs to be solved; hence this process is referred to as *maintenance*. There can be many reasons for this change to occur. Change could happen because of some unexpected input values into the system. Thus, maintenance includes enhancements, modifications or any change from the original specifications.

2.4 THE NEED FOR A SOFTWARE LIFE CYCLE MODEL

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

2.5 SYSTEM DEVELOPMENT APPROACHES

In order to make sure that the systems are analysed and designed efficiently and effectively, it is essential to adopt a suitable system development model. Typically, a system development model specifies how the activities are organised in the total development effort. It is a descriptive and diagrammatic representation of the *System Development Life Cycle*.

Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. The models presented in the following sections are the most

common system development life cycle models. In this unit we will discuss only three important life cycle model. These are : The waterfall model, the prototype model and the Sprial model.

2.5.1 The Waterfall Model

The *waterfall model,* which follows the SDLC(System Development Life Cycle) approach is a popular version of the system development life cycle model although it is the oldest paradigm for software engineering. This model is sometimes called the *linear sequential model*. There are many versions of waterfall model. Although the specific development tasks will occur in almost every development, the waterfall model describes development method as linear and sequential.

In the waterfall approach, the whole process of software development is divided into separate phases/stages with distinct goals. The relationship of each phase to the others can be roughly described as a waterfall, where the outputs from a specific stage serve as the initial inputs for the following stage. The phases in waterfall model are depicted in figure 2.1.

Advantages of Waterfall Model :

- It is simple to implement
- It is systematic and sequential
- It has proper documentation

Disadvantages of the Waterfall Model :

- All requirements are not received at once. The requirements from customer go on getting added to the list even after the end of “Requirement analysis” phase. This affects the system development process and its success in negative aspects.
- This model is not suitable for accommodating any change.
- The developer cannot go in the backward direction once the life cycle begins. If there is any mistake or error in any phase, then this results in the development of on almost unusable system.

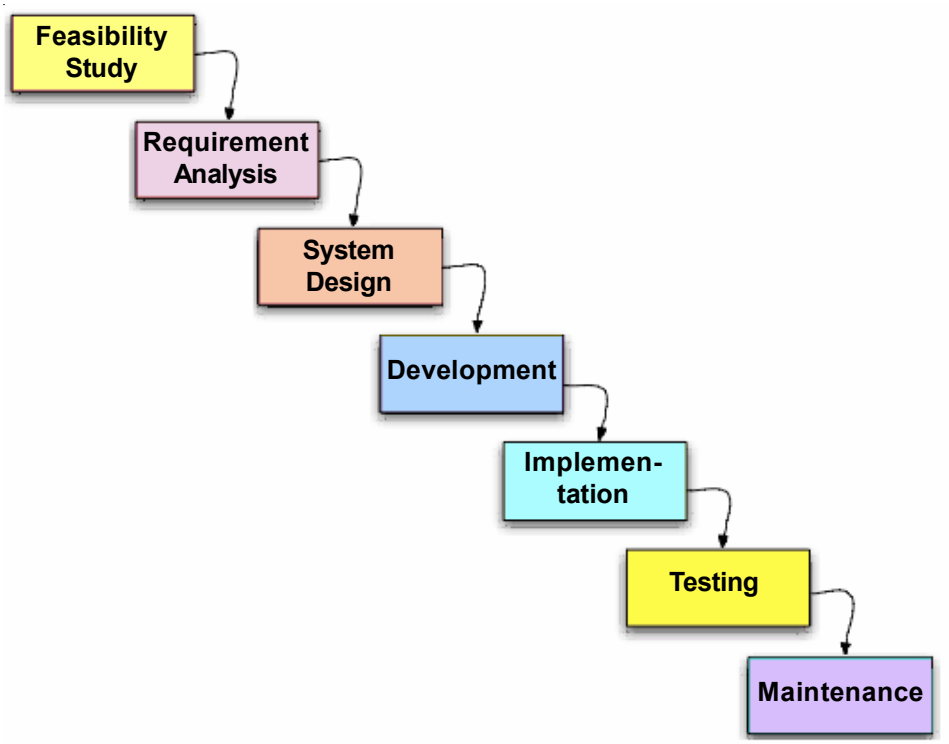


Figure 2.1: The Waterfall Model

Applicability of Waterfall Model

- When requirements are well known and a few changes are likely to be needed.
- Can be used also for parts of larger software systems.

2.5.2 The Prototyping Model

A prototype ia a comprehensive system and does not include all the details of the system. In the prototyping model, once the requirement analysis is done and the design for a prototype is made, the development process gets started. Once the prototype is created, it is given to the customer for evaluation. The customer tests the package and gives his/her feedback to the developer who refines the product according to the customer's exact expectation. After a finite number of iterations, the final software package is given to the customer. This model is based on the evolutionaly method of system development. The model, in general, has the following four steps:

- Identify the user's basic information requirements
- Develop the initial prototype system to meet the basic requirements
- Use of the prototype system to refine the user's requirements

- Revise and enhance the prototype system.

In this methodology, the software is evolved as a result of periodic shuttling of information between the customer and the developer. Most of the successful software products have been developed using this model - as it is very difficult to comprehend all the requirements of a customer in advance. The principal use is to help the customers and the developers to understand the requirements for the system. Prototyping model is becoming increasingly used for system development where rapid development is essential. It is used in those systems, in which identification of the requirements is difficult and requirements may change during the development process.

Advantages of Prototyping Model :

- The ability to get a functioning system into the hands of the user quickly.
- Early visibility of the prototype gives users an idea of what the final system looks like.
- Feedback from customers are received periodically and the changes donot come as a last moment of development.

Disadvantages of Prototyping Model :

- Not suitable for large applications
- Can cause systems to be left unfinished and/or implemented before they are ready.

2.5.3 The Spiral Model

The spiral model was defined by *Barry Boehm* in his article *"A Spiral Model of Software Development and Enhancement"* in1985. This model is designed to include the best features from the *waterfall* and *prototyping models* and it introduces a new component-risk assessment. Similar to the prototyping model, an initial version of the system is developed and then repetitively modified based on input received from customer evaluations. Unlike the prototyping model, however, the development of each version of the system is carefully designed using the steps involved in the waterfall model.

As the name suggests, the activities in this model can be organized like a spiral as shown in figure 2.2. The spiral has many cycles. Each cycle of the spiral consists of four stages represented by one quadrant each. The angular dimension represents the progress in the development process, whereas the radial dimension represents the cumulative cost incurred in accomplishing the steps done so far. Each cycle in the spiral begins with the identification of the objectives for that cycle and the different alternatives are possible for achieving

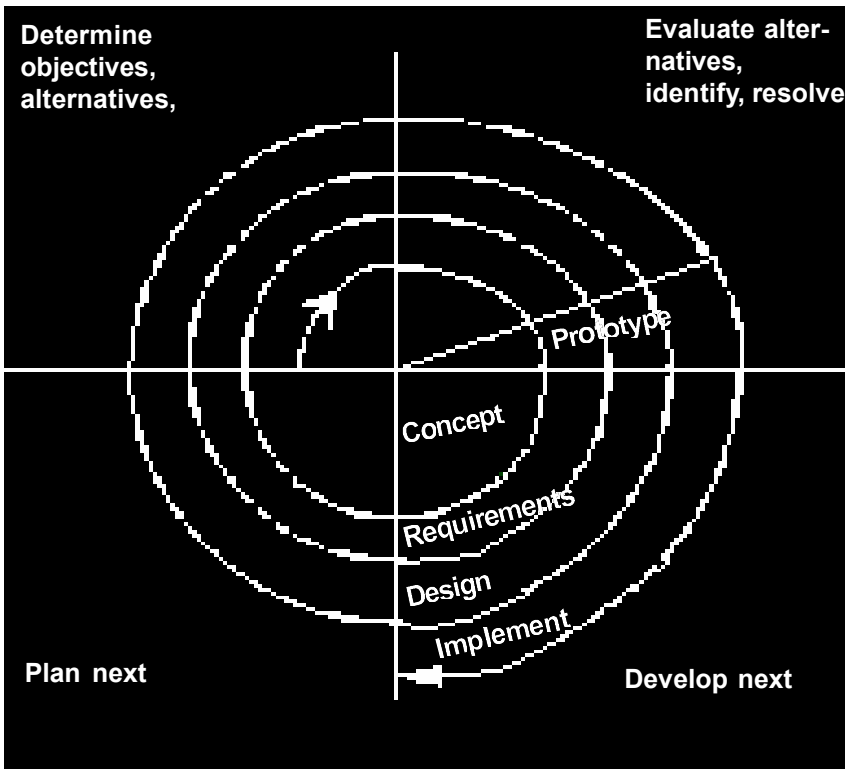


Figure 2.2: Spiral Model

the objectives and the imposed constraints. The evaluation of various alternatives and identification of risks and uncertainities are undertaken in the second stage. In the third stage, the next level prototype is developed and verified. The result of the previous stages are reviewed and planning for the next iteration is done in the fourth stage. An important feature of the model is that each cycle of the spiral is completed by a review, which covers all the products developed during that cycle, including plans for the next cycle.

The key characteristic of a spiral model is risk assessment at regular stages in the development cycle. Risk assessment is included at each stage of the development process as a means of evaluating each version of the system to determine whether or not development should continue. If the customer decides that any identified risks are too high, the project may be halted. For example, if a substantial increase in cost or project completion time is identified during one phase of risk assessment, the customer or the developer may decide that it does not make sense to continue with the project, since the increased cost or lengthened timeframe may make continuation of the project impractical or unfeasible.

Advantages of Spiral Model :

- The spiral model is suitable for the development of high-risk large-scale systems and softwares. Because software evolves as process progresses, the developer and the customer better understand and react to risks at

each evolutionary level.

- The spiral model uses prototyping as a risk reduction mechanism but, more importantly, enables the developer to apply the prototyping approach at any stage in the evolution of the product.
- Users see the system early because of rapid prototyping tools.

Disadvantages of Spiral Model :

- Requires considerable expertise in risk evaluation and reduction.
- For small project, this model may not be time- and cost-effective.

We have discussed three different approaches to software development. Depending upon the nature and size of the project and risk involved therein, a combination of more than one model may be an appropriate strategy.

CHECK YOUR PROGRESS

Q1. Fill in the blanks :

(i) In system development, the total development process is devided into smaller activities called _____.

(ii) In the waterfall model, every phase is considered as a _____ phase.

(iii) _____ model is used in those systems in which identification of the requirements is difficult and requirements may change during the development process.

(iv) _____ model is based on the evolutionaly method of system development.

Q2. State True or False:

(i) Waterfall model states that the phases are organized in a linear order.

(ii) The waterfall model is more suitable for high-risk project.

(iii) For small project, spiral model may not be time and cost effective.

(iv) Prototyping model lowers the overall development cost when requirements change frequently.

2.6 LET US SUM UP

- System development is regarded as a complete process consisting of various phases, namely, *feasibility study*, *analysis*, *design*, *development*, *implementation*, *testing and maintenance*.
- There are several development approaches which could be adopted in developing an information system.

- Three approaces:***waterfall model***, ***prototyping model*** and ***spiral model*** are presented with their relative strengths and weaknesses.
- No approach could be termed as the best approach.
- It is the developer of the system, who decides a particular approach or a combination of more than one approach for developing a system, depending on the situation.

2.7 FURTHER READINGS

- Jawdekar, W. S. (2004). Software Engineering Principles and Practices.
- Leach, R. J. (2016). Introduction to software engineering. CRC Press.
- Mall, R. (2014). Fundamentals of software engineering. PHI Learning Pvt. Ltd.

2.8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: (i) phase/stage,
(ii) distinct,
(iii) Prototyping,
(iv) Prototyping

Answer to Q2: (i) True,
(ii) False,
(iii) True,
(iv) True

2.9 MODEL QUESTIONS

Q1. Briefly explain the phases involved in system development.

Q2. What is a system prototype? Will you recommend the use of prototype method for system development? When and why?

Q3. What is feasibility study?

Q4. What are the merits and demerits of Waterfall model of system development?

Q5. Write short notes on any one of the following:
(a) Prototyping Model
(b) Spiral Model
(c) Waterfall model

Q6. Discuss the relative advantages and disadvantages of prototyping and spiral model.

****

UNIT 3: SOFTWARE PROJECT MANAGEMENT

UNIT STRUCTURE

3.1 Learning Objectives
3.2 Introduction
3.3 Software Project Manager
3.4 Project Planning
- 3.4.1 Project Planning Process
- 3.4.2 Scope Management
- 3.4.3 Project Estimation
- 3.4.4 Metrics for Software Project Size Estimation

3.5 Project Estimation Techniques
3.6 Empirical Estimation Techniques
3.7 Heuristic Techniques
3.8 COCOMO Model
- 3.8.1 Organic, Semidetached and Embedded Software Projects
- 3.8.2 COCOMO
- 3.8.3 Basic COCOMO Model
- 3.8.4 Intermediate COCOMO Model
- 3.8.5 Complete COCOMO Model

3.9 Analytical Estimation Techniques
- 3.9.1 Halstead's Software Science

3.10 Risk Management
- 3.10.1 Project Risk Management

3.11 Configuration Management
3.12 Let Us Sum Up
3.13 Further Readings
3.14 Answers to Check Your Progress
3.15 Model Questions

3.1 LEARNING OBJECTIVES

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

- learn project planning
- describe different project estimation techniques
- describe COCOMO model
- describe analytical heuristic model
- define risk and configuration management

3.2 INTRODUCTION

We are now familiar with software development and its methodologies. We have already studied the importance of software engineering and different system development models. In this unit, we will learn about software project management, Project planning and its different techniques. Software project management is a process of managing and allocating resources to develop software that meets the requirements of the user. It allows project managers, stakeholders and users to control costs and manage budgeting, quality management, documentation and it may also be used as an administration system. In software project management the end users and developers need to know the length, duration and cost of the project. Therefore, software project management is essential to incorporate the user requirements along with budget and time constraints. Planning is very important for software development. Without proper planning there may be slippage of schedule. We need to use different techniques and tools depending on project parameter such as size, time, cost etc. and these will be monitored and controlled by software project manager. Once we are familiar with software project management in this units, then in the next unit, we will learn about the issues related to staffing and scheduling.

3.3 SOFTWARE PROJECT MANAGER

A software project manager is a person who defines the requirements of the project, builds the project team, lays out a blue print for the whole project including the project scope and parameters. He communicates the goals of the project to the project team and the targets to be achieved he/she allots budget to the various tasks to be completed and ensures that the expectations of the clients are met through timely completion of the project.

It is very difficult to list the responsibilities of a project manager. The specific **duties of a project manager** vary from industry to industry, company to company, and sometimes even from project to project. The job responsibility of a project manager ranges from invisible activities like building up team morale to highly visible customer presentations. Some of the responsibilities are listed below:

- Planning and defining scope
- Activity planning and sequencing
- Resource planning
- Developing schedules
- Time estimating
- Cost estimating

- Developing a budget
- Documentation
- Creating charts and schedules
- Risk analysis
- Managing risks and issues
- Monitoring and reporting progress
- Team leadership
- Controlling quality

Software Management Activities

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

- Project planning
- Scope management
- Project estimation

3.4 PROJECT PLANNING

It is essential to determine the tasks to be performed and manage properly the allocation of tasks among the individuals involved in the software development before starting a software project. Hence, for effective software development, project planning is important.

Project planning is an organized and integrated management process which focuses on the activities required for successful completion of the project. It prevents obstacles that may arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

- It defines the roles and responsibilities of the project management team members.
- It ensures that the project management team works according to the business objectives.
- It checks the feasibility of the schedule and user requirements.
- It determines project constraints.

3.4.1 Project Planning Process

Project planning process consists of the following activities.

- **Identification of project requirements:** Before starting a project, it is essential to identify the project requirements as identification of project requirements helps in performing the activities in a systematic manner. These requirements comprise information such as project scope, data and functionality required in the software, and roles of the project management team members.
- **Identification of cost estimates:** Along with the estimation of effort and time, it is necessary to estimate the cost that is to be incurred on a project. The cost estimation includes the cost of hardware, network connections, and the cost required for the maintenance of hardware components. In addition, cost is estimated for the individuals involved in the project.
- **Identification of risks:** Risks are unexpected events that have an adverse effect on the project. Software project involves several risks (like technical risks and business risks) that may affect the project schedule and increase the cost of the project. Identifying risks before a project begins helps in understanding their probable extent of impact on the project.
- **Identification of critical success factors:** For making a project successful, critical success factors are followed. These factors refer to the conditions that ensure greater chances of success of a project. Generally, these factors include support from management, appropriate budget, appropriate schedule, and skilled software engineers.
- **Preparation of project charter:** A project charter provides a brief description of the project scope, quality, time, cost, and resource constraints as described during project planning. It is prepared by the management for approval from the sponsor of the project.
- **Preparation of project plan:** A project plan provides information about 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:** Once the project planning is complete and resources are assigned to team members, the software project commences.

3.4.2 Scope Management

Scope management is essential because it clearly defines what would be done in the project and what would not be done. This makes project a contain limited and quantifiable

tasks which can easily be documented and the in turn can avoid cost and time overrun. During project scope management, we need to define and verify the scope, divide the project into various smaller parts for ease of management and control the scope by incorporating changes to the scope.

3.4.3 Project Estimation

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable. For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

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

- **Effort estimation**

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

- **Time estimation**

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

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

- **Cost estimation**

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

  - Size of software
  - Software quality

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

3.4.4 Metrics for Software Project Size Estimation

There are two metrics popularly being used widely to estimate the size:

Line of Code: Estimation is done on behalf of number of line of codes in the software product.

Function Points: Estimation is done on behalf of a number of function points in the software product.

CHECK YOUR PROGRESS

Q1. Write down the activities of Project Planning Process.

Q2. List out the important responsibilities of Software Project Manager.

Q3. What is Scope Management?

Q4. Which are the matrices used for Project Software Size Estimation?

Q5. State whether the following statements are True(T) or False(F).

(i) Software project estimation can never be an exact science, but a combination of good historical data and systematic techniques can improve estimation accuracy.

(ii) Software Project management goal is avoiding customer complaints.

(iii) Quality planning is the process of developing a quality plan for Projects.

3.5 PROJECT ESTIMATION TECHNIQUES

Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated includes project size, effort required to develop the software, project duration, and cost. These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling. There are three broad categories of estimation techniques:

- Empirical estimation techniques
- Heuristic techniques
- Analytical estimation techniques

3.6 EMPIRICAL ESTIMATION TECHNIQUES

Empirical estimation techniques are based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is required. Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are: Expert judgment technique and Delphi cost estimation.

Expert Judgment Technique

Expert judgment is one of the most widely used estimation technique. In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly. Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate. However, this technique is subject to human errors and individual bias. Also, it is possible that the expert may overlook some factors inadvertently. Further, an expert making an estimate may not have experience and knowledge of all aspects of a project. For example, he may be conversant with the database and user interface parts but may not be very knowledgeable about the communication and networking part.

A more refined form of expert judgment is the estimation made by group of experts. Estimation by a group of experts minimizes factors such as individual oversight, lack of familiarity with a particular aspect of a project, personal bias, and the desire to win contract through overly optimistic estimates. However, the estimate made by a group of experts may still exhibit bias on issues where the entire group of experts may be biased due to reasons such as political considerations. Also, the decision made by the group may be dominated by overly assertive members.

Delphi cost estimation

Delphi cost estimation approach tries to overcome some of the shortcomings of the expert judgment approach. Delphi estimation is carried out by a team comprising a group of experts and a coordinator. In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for cost estimate. Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation. The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators. Based on this summary, the estimators re-estimate the cost. This process is iterated for several rounds. However, no discussion

among the estimators is allowed during the entire estimation process. The idea behind this is that if any discussion is allowed among the estimators, then many estimators may easily get influenced by the rationale of an estimator who may be more experienced or senior. After the completion of several iterations of estimations, the coordinator takes the responsibility of compiling the results and preparing the final estimate.

3.7 HEURISTIC TECHNIQUES

Heuristic techniques assume that the relationships among the different project parameters can be modelled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters. Different heuristic estimation models can be divided into the following two classes:

- Single Variable Model
- Multi Variable Model

Single Variable Model

Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size. A single variable estimation model takes the following form:

$$\text{Estimated Parameter} = c1 * e^{d1}$$

In the above expression, e is the characteristic of the software which has already been estimated (independent variable). Estimated Parameter is the dependent parameter to be estimated which could be effort, project duration, staff size, etc. c1 and d1 are constants. The values of the constants c1 and d1 are usually determined using the data collected from past projects (historical data). The basic COCOMO model is an example of single variable cost estimation model.

Multi variable model

$$\text{Estimated Resource} = c1*e1^{d1} + c2*e2^{d2} + \ldots$$

where e1 , e2 , … are the basic (independent) characteristics of the software already estimated, and c1 , c2 , d1 , d2 , … are constants. Multivariable estimation models are expected to give more accurate estimates compared to the single variable models. The independent parameters influence the dependent parameter to different extents. This is modelled by the constants c1, c2, d1, d2, … . Values of these constants are usually

determined from historical data. The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.

3.8 COCOMO MODEL

3.8.1 ORGANIC, SEMIDETACHED AND EMBEDDED SOFTWARE PROJECTS

Organic

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 types of projects.

Semidetached

If the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded

If the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

3.8.2 COCOMO

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm in 1981. According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

3.8.3 BASIC COCOMO MODEL

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

$$\text{Effort} = a_1 \times (\text{KLOC})^{a_2} \text{ PM}$$

$$\text{Tdev} = b_1 \times (\text{Effort})^{b_2} \text{ Months}$$

Where

- KLOC is the estimated size of the software product expressed in Kilo lines of Code,
- a_1, a_2, b_1, b_2 are constants for each category of software products,
- Tdev is the estimated time to develop the software, expressed in months,

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

Estimation of development effort

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

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

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

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

Estimation of development time

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

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

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

Embedded: Tdev = $2.5(Effort)^{0.32}$ Months

Example 3.1: Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

From the basic COCOMO estimation formula for organic software:

$$\text{Effort} = 2.4 \times (32)^{1.05} = 91 \text{ PM}$$

Nominal development time = $2.5 \times (91)^{0.38}$ = 14 months

Cost required to develop the product = 14 õ 15,000

= Rs. 210,000/-

3.8.4 Intermediate COCOMO model

The basic COCOMO model assumes that effort and development time are functions of the product size alone. However, a host of other project parameters besides the product size affect the effort required to develop the product as well as the development time. Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account. The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development. For example, if modern programming practices are used, the initial estimates are scaled downward by multiplication with a cost driver having a value less than 1. If there are stringent reliability requirements on the software product, this initial estimate is scaled upward. Boehm requires the project manager to rate these 15 different parameters for a particular project on a scale of one to three. Then,

depending on these ratings, he suggests appropriate cost driver values which should be multiplied with the initial estimate obtained using the basic COCOMO. In general, the cost drivers can be classified as being attributes of the following items:

Product: The characteristics of the product that are considered include the inherent complexity of the product, reliability requirements of the product, etc.

Computer: Characteristics of the computer that are considered include the execution speed required, storage space required etc.

Personnel: The attributes of development personnel that are considered include the experience level of personnel, programming capability, analysis capability, etc.

Development Environment: Development environment attributes capture the development facilities available to the developers. An important parameter that is considered is the sophistication of the automation (CASE) tools used for software development.

3.8.5 Complete COCOMO model

A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics. For example, some sub-systems may be considered as organic type, some semidetached, and some embedded. Not only that the inherent development complexity of the subsystems may be different, but also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development, and so on. The complete COCOMO model considers these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems. The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

The following development project can be considered as an example of the application of the complete COCOMO model. A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- Database part
- Graphical User Interface (GUI) part
- Communication part

Of these, the communication part can be considered as embedded software. The database part could be semi-detached software, and the GUI part organic software. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.

CHECK YOUR PROGRESS

Q6. What are the three main project estimation techniques?

Q7. What is the estimated parameter of single variable model of heuristic techniques?

Q8. Multiple Choice Questions:

(i) A COCOMO model is _____ .

a) Common Cost Estimation Model.
b) Constructive Cost Estimation Model.
c) Complete Cost Estimation Model.
d) Comprehensive Cost Estimation Model

(ii) COCOMO II is an example of a suite of modern empirical estimation models that require sizing information expressed as:

a) Function points
b) Object points
c) Lines of Code
d) Any of the above

(iii) The model which estimates the total effort in terms of person, months of the technical project staff is _____ .

a) Spiral Model
b) Waterfall model
c) Win-win spiral model
d) Cocomo Model

3.9 ANALYTICAL ESTIMATION TECHNIQUES

Analytical estimation techniques derive the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. Halstead's software science is an example of an analytical technique.

- It can be used to derive some interesting results starting with a few simple assumptions.
- Halstead's software science is especially useful for estimating software maintenance efforts. In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.

3.9.1 Halstead's Software Science

Halstead's software science is an analytical technique to measure size, development effort, and development cost of a software product. Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum value, actual volume, effort, and development time. For a given program, let:

- η_1 be the number of unique operators used in the program,
- η_2 be the number of unique operands used in the program
- N_1 be the total number of operators used in the program,
- N_2 be the total number of operands used in the program.

Different Parameters

Length and Vocabulary

- The length of a program is total usage of all operators and operands in the program. Thus, length $N = N_1 + N_2$.
- program vocabulary is the number of unique operators and operands used in the program. Thus, program vocabulary $\eta = \eta_1 + \eta_2$.

Program Volume

- $V = N\log_2 \eta$
- Here the program volume V is the minimum number of bits needed to encode the program.

Potential Minimum Value:

The potential minimum volume V* is defined as the volume of most succinct program in which a problem can be coded. The minimum volume is obtained when the program can be expressed using a single source code instruction, say a function call like foo( ) ;. In other words, the volume is bound from below due to the fact that a program would have at least two operators and no less than the requisite number of operands. Thus, if an algorithm operates on input and output data d1 , d2 , … d n , the most succinct program would be $f(d^1 , d^2 , \ldots d^n)$; for which $\eta_1 = 2$, $\eta_2 = n$. Therefore, $V^* = (2 + \eta_2) \log_2 (2 + \eta_2)$.

The program level L is given by $L = V^*/V$. The concept of program level L is introduced in an attempt to measure the level of abstraction provided by the programming language. Using this definition, languages can be ranked into levels that also appear intuitively correct. The above result implies that the higher the level of a language, the less effort it takes to develop a program using that language. This result agrees with the intuitive notion that it takes more effort to develop a program in assembly language than to develop a program in a high-level language to solve a problem.

Effort and Time:

The effort required to develop a program can be obtained by dividing the program volume with the level of the programming language used to develop the code. Thus, effort E = V/L, where E is the number of mental discriminations required to implement the program and also the effort required to read and understand the program. Thus, the programming effort E = V²/V* (since L = V*/V) varies as the square of the volume. Experience shows that E is well correlated to the effort needed for maintenance of an existing program. The programmer's time T = E/S, where S the speed of mental discriminations. The value of S has been empirically developed from psychological reasoning, and its recommended value for programming applications is 18.

Length Estimation:

In terms of unique operators and operands

- It can be assumed that any program of length N consists of N/ η unique strings of length η.
- Now, it is standard combinatorial result that for any given alphabet of size K, there are exactly K^r different strings of length r.

Thus, ‼ N/η d" η^{η} Or, N d" $\eta^{\eta+1}$

Since operators and operands usually alternate in a program, the upper bound can be further refined into N d" $\eta\ \eta_1^{\eta 1}\ \eta_2^{\eta 2}$. Also, N must include not only the ordered set of n elements, but it should also include all possible subsets of that ordered sets, i.e. the power set of N strings

Therefore,

2 N = $\eta\ \eta_1^{\eta 1}\ \eta_2^{\eta 2}$ Or, taking logarithm on both sides,

N = $\log_2\eta + \log_2(\eta_1^{\eta 1}\ \eta_2^{\eta 2})$

So we get, N = $\log_2(\eta_1^{\eta 1}\ \eta_2^{\eta 2})$approx by ignoring $\log_2\eta$

N= $\eta_1 \log_2\eta_1 + \eta_2 \log_2\eta_2$

In conclusion, Halstead's theory tries to provide a formal definition and quantification of such qualitative attributes as program complexity, ease of understanding, and the level of abstraction based on some low-level parameters such as the number of operands, and operators appearing in the program. Halstead's software science provides gross estimation of properties of a large collection of software, but extends to individual cases rather inaccurately.

Example 3.2: Let us consider the following C program:

```
main()
{
      int  a,b,c,avg;
```

```
        scanf("%d%d%d", &a, &b, &c);
        avg = (a+b+c)/3;
        printf("avg = %d", avg);
    }
```

The unique operators are:

main,(),{},int,scanf,&,",",";",=,+,/, printf

The unique operands are:

a, b, c, &a, &b, &c, a+b+c, avg, 3,

"%d %d %d", "avg = %d

Therefore,

$\eta_1 = 12$, $\eta_2 = 11$

Estimated Length = (12*log12 + 11*log11)

= (12*3.58 + 11*3.45)

= (43+38) = 81

Volume = Length*log(23)

= 81*4.52

= 366

3.10 RISK MANAGEMENT

Risk concerns future happenings. It is inevitable in a business organization when undertaking projects. However, the project manager needs to ensure that risks are kept to a minimal. Risks can be mainly divided into two types, negative impact risk and positive impact risk.

The project manager would not all the time face the negative impact risks as there are positive impact risks too. Once the risk has been identified, the project manager need to come up with a mitigation plan or any other solution to counterattack the risk.

3.10.1 Project Risk Management

Managers can plan their strategy based on four steps of risk management which prevails in an organization. Following are the steps to manage risks effectively in an organization:

a) Risk Identification

b) Risk Quantification

c) Risk Response

d) Risk Monitoring and Control

Let's go through each of the step in project risk management:

Risk Identification

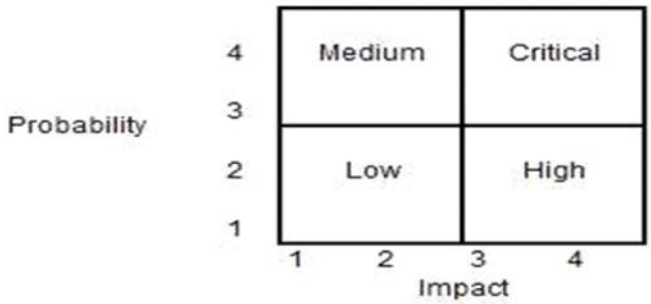
Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading etc). By identifying known and predictable risks, the project manager takes the first steps towards avoiding them when possible. Managers face many difficulties when it comes to identifying and naming the risks that occur when undertaking projects. These risks could be resolved through structured or unstructured brainstorming or strategies. It is important to understand that risks pertaining to the project can only be handled by the project manager and other stakeholders of the project.

Risks, such as operational or business risks will be handled by the relevant teams. The risks that often impact a project are supplier risk, resource risk and budget risk. Supplier risk would refer to the risks that can occur in case the supplier is not meeting the timeline to supply the resources required.

Resource risk occurs when the human resource used in the project is not enough or not skilled enough. Budget risk would refer to risks that can occur if the costs are more than what was budgeted.

Risk Quantification

Risks can be evaluated based on quantity. Project managers need to analyze the likely chances of a risk occurring with the help of a matrix.



Using the matrix, the project manager can categorize the risk into four categories as Low, Medium, High and Critical. The probability of occurrence and the impact on the project are the two parameters used for placing the risk in the matrix categories. As an example, if a risk occurrence is low (probability = 2) and it has the highest impact (impact = 4), the risk can be categorized as ‘High’.

Risk Response

When it comes to risk management, it depends on the project manager to choose strategies that will reduce the risk to minimal. Project managers can choose among the four risk response strategies, which are outlined below.

- Risks can be avoided
- Pass on the risk

- Take corrective measures to reduce the impact of risks
- Acknowledge the risk

Risk Monitoring and Control

Risks can be monitored on a continuous basis to check if any change is made. New risks can be identified through the constant monitoring and assessing mechanisms.

Risk Management Process

Following are the considerations when it comes to risk management process:

- Each person involved in the process of planning needs to identify and understand the risks pertaining to the project.
- Once the team members have given their list of risks, the risks should be consolidated to a single list in order to remove the duplications.
- Assessing the probability and impact of the risks involved with the help of a matrix.
- Split the team into subgroups where each group will identify the triggers that lead to project risks.
- The teams need to come up with a contingency plan whereby to strategically eliminate the risks involved or identified.
- Plan the risk management process. Each person involved in the project is assigned a risk in which he/she looks out for any triggers and then finds a suitable solution for it.

3.11 CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) is the discipline for systematically controlling the changes that take place during the development process. Software configuration management is a process independent of the development process largely because most development models cannot accommodate change at any time during development.

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

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

Baseline

A phase of system development life cycle (SDLC) is assumed over if it is baselined, where baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not

the final phase, its output would be used in the next immediate phase.

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

Change Control

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

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

- Identification - A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- Validation - Validity of the change request is checked and its handling procedure is confirmed.
- Analysis - The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- Control - If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.
- Execution - If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change and does a thorough revision if necessary.
- Close request - The change is verified for correct implementation and merges with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

CHECK YOUR PROGRESS

Q9. What is analytical estimation technique?

Q10. What is change control?

Q11. State whether the following statements are true(T) or false(F).

i) Length Estimation using Halstead's Theory is $N= \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$, where N is the Length of Program ,η_1 is the number of unique operators used in the program and η_2 is the number of unique operands used in the program.

ii) Every time project managers will be facing negative impact risks.

iii) Halstead Software Science can be used to derive some interesting results starting with a few simple assumptions.

3.12 LET US SUM UP

- A software project manager is a person who undertakes the responsibility of executing the software project.
- Project planning is a discipline for stating how to complete a project within a certain timeframe, usually with defined stages and with designated resources.
- Scope management is the process whereby outputs, outcomes and benefits are identified, defined and controlled.
- Project Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.
- There are mainly three different project estimation techniques – Empirical estimation techniques, Heuristic Techniques and Analytical estimation techniques.
- Empirical estimation techniques are based on common sense such as an expert's intellegent guess etc.
- Heuristic techniques are used to express the relationships among the different project parameters using suitable mathematical expressions.
- The intermediate COCOMO model recognizes the effect of all relevant parameters of the project and refines the initial estimate obtained using the basic COCOMO expressions by using a set of cost drivers based on various attributes of software development.
- The complete COCOMO model considers all the differences in characteristics of the subsystems of main the systems and estimates the effort and development time as the sum of the estimates for the individual subsystems
- *Risk management is the process of identifying risk, assessing risk, and taking steps to reduce risk to an acceptable level*.
- The steps to manage risks effectively in an organization are Risk Identification, Risk Quantification, Risk Response, *Risk Monitoring and Control.*
- *Software Configuration Management involves identifying configuration items for the software project, controlling these configuration items and changes to them, and recording and reporting status and change activity for these configuration items*.

3.13 FURTHER READINGS

- Mall, R. (2014). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd.

- Leach, R. J. (2016). *Introduction to software engineering*. CRC Press.
- Jalote, P. (2012). *An integrated approach to software engineering*. Springer Science & Business Media

✓ 3.14 ANSWERS TO THE CHECK YOUR PROGRESS

Answer to Q1: Project planning process consists of the following activities:

a) Identification of critical success factors
b) Identification of critical success factors
c) Identification of project requirements
d) Identification of cost estimates
e) Identification of risks
f) Preparation of project charter
g) Preparation of project plan
h) Commencement of the project

Answer to Q2: Some of the important responsibilities of software project manager are listed below:

a) Planning and Defining Scope
b) Activity Planning and Sequencing
c) Resource Planning
d) Developing Schedules
e) Time Estimating
f) Cost Estimating
g) Developing a Budget

Answer to Q3: Scope Management defines what need to be done in the project and what would not be done, This makes project to contain limited and quantifiable tasks, which can easily be documented and the project in turn, avoids cost and time overrun.

Answer to Q4: There are two metrics that are used to estimate size-

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

Answer to Q5: (i) T (ii) F (iii) T

Answer to Q6: There are main estimation techniques:

a) Empirical estimation techniques,
b) Heuristic techniques and
c) Analytical estimation techniques

Answer to Q7: Estimated Parameter = c1* ed1 where e is the characteristic of the

software which has already been estimated and c1, d1 are constants.

Answer to Q8: (i) b (ii) d (iii) d

Answer to Q9: Analytical estimation techniques derive the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. Halstead's software science is an example of an analytical technique.

Answer to Q10: Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

Answer to Q11: (i) T (ii) F (iii) T

3.15 MODEL QUESTIONS

Q1. What are the differences between cost estimation and time estimation?

Q2. Explain project estimation and its types.

Q3. What are the two models of heuristic techniques?

Q4. What is Work Breakthrough Structure (WBS)?

Q5. What are the main objectives of project planning?

Q6. Explain Intermediate COCOMO Model. How does it differ from the basic model?

Q7. What are Organic, Semidetached and Embedded software projects?

Q8. Explain complete COCOMO model with an example.

Q9. Write down the steps of project risk management.

Q10. How does matrix help the project manager in risk quantification?

Q11. What are the four risk response strategies a project manager can choose?

Q12. Explain software configuration management.

UNIT 4: STAFFING AND SCHEDULING

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 Project Staffing

- 4.3.1 Team Structure
- 4.3.2 Chief Programmer Team
- 4.3.3 Egoless Team
- 4.3.4 Controlled Decentralized Team
- 4.3.5 Characteristic of a good Software Engineer

4.4 Project Scheduling

- 4.4.1 Work Breakdown Structure
- 4.4.2 Activity Networks
- 4.4.3 Critical Path Method
- 4.4.4 PERT Charts
- 4.4.5 Gantt Charts

4.5 Let Us Sum Up

4.6 Further Readings

4.7 Answer to Check Your Progress

4.8 Model Questions

4.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- describe the importance of project staffing
- classify different project team structures
- define different project cost estimation methods
- understand the needs of project scheduling
- define the importance of critical path method
- learn different project scheduling methods
- differentiate the objectives of Gantt charts and PERT charts

4.2 INTRODUCTION

In the previou few units, we have learnt about the different software life cycle models. We have also learnt about the different concepts related to software project management. Different estimation methods like heuristic, empirical etc were covered in detail in the previous unit. In this unit we, we will learn about staffing and scheduling concepts.

Software staffing, cost-estimation and project scheduling are very important steps in software planning and management activity. The proper and cost effective software product depends on proper staff organization, team structures. The different types of models are available for estimating project cost which can make the software budget efficient and cost effective. Project scheduling is another important activity which plays an important role in proper execution of software leading to achieving the required milestone in the software project management plan. A milestone is a predefined goal which is expected to be achieved during the project plan and it indicates how far the project has progressed.

4.3 PROJECT STAFFING

Project staffing is important in software scheduling which defines the organization of different project teams used in scheduling. It also includes the identification of the right person for handling the project, assignment of required responsibilities to the selected team. Different projects may require different skilled and knowledge the individuals as well as team scheduling. A software project manager usually takes the responsibility of accomplishing the software project. He needs to identify good, experienced software engineers for the project. He is also accountable of choosing their team with proper organization. The proper selection of software engineers reflects the overall productivity of the entire software team.

4.3.1 Team Structure

The structure of working team during the development process is very important. The team structure means thearrangement of the individual project teams. Project team must be arranged in such a way that they work in an organized manner and complete the project as a unit. There are several possible ways in which the individual project teams can be organized. The team structure always makes a desired impact on the productivity and quality of product. Based on the type and complexity of the project the project manager needs to organize his team in different ways. There are basically three formal team structures: chief programmer team, egoless teamand controlled decentralized teamorganizations.

4.3.2 Chief Programmer Team

In this team structure, there is a senior engineer who affords the technical leadership for the project. He is designated as the chief programmer of the team. He makes the partitions of the task into many smaller tasks and allocates them to different team members under his leadership. He is the only one person who will communicate with their project members. Any other project member except him has no authority to communicate with

each other during the working of the project task. He is also responsible for verification and integration of the products developed by its different team members. The structure of the chief programmer team is shown in figure 4.1.

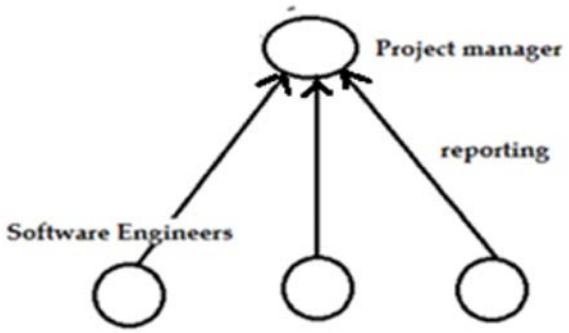


Figure 4.1: Chief programmer team

This team structure is most competent when the work is small and when the task is well understood. However, this team structure may lead to lower team morale because of the constant supervision of the chief. The original assessment of the team members may also be suppressed. This team structure is subject to single point failure since the chief programmer may either leave the organization or becomes unavailable for some reasons during the progress of the task.

4.3.3 Egoless Team

The egoless team structurevery simple and does not follow any formal team hierarchy as shown in figure 4.2. The main aim is to involve each and every member of the team in any decision making during the development process of the proposed product. Typically, a manager provides the administrative leadership.

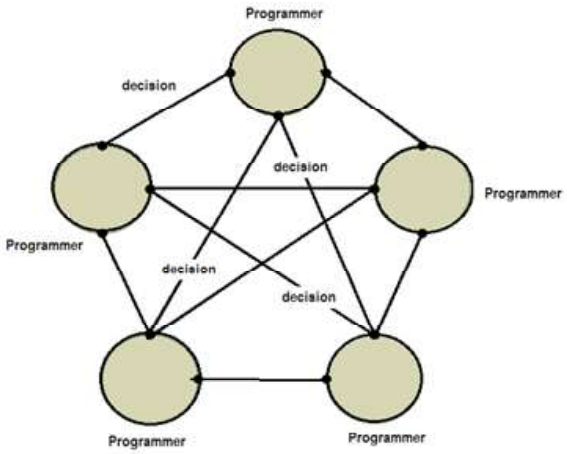


Figure 4.2: Egoless team structure

Different members of the group may provide technical leadership in different time. Here, decisions are made by common agreement among the team members. In this structure every member's suggestions are taken into account. Due to this characteristic, it is also often termed as democratic team structure. This team structure is suitable for handling complex and difficult project with many technical challenges.

4.3.4 Controlled Decentralized Team

The controlled decentralized team mixes the ideas from both the egoless organization and the chief-programmer team organization. Controlled decentralized team organization is shown in fig. 4.3. This team structure consists of a project leader, who has a group of senior programmer. Each of the senior programmers again has a group of junior programmer under his leadership. The project leader communicates with the senior programmers and the senior programmers communicate with the junior programmers. The senior leaders are not allowed to communicate among themselves for decision making.

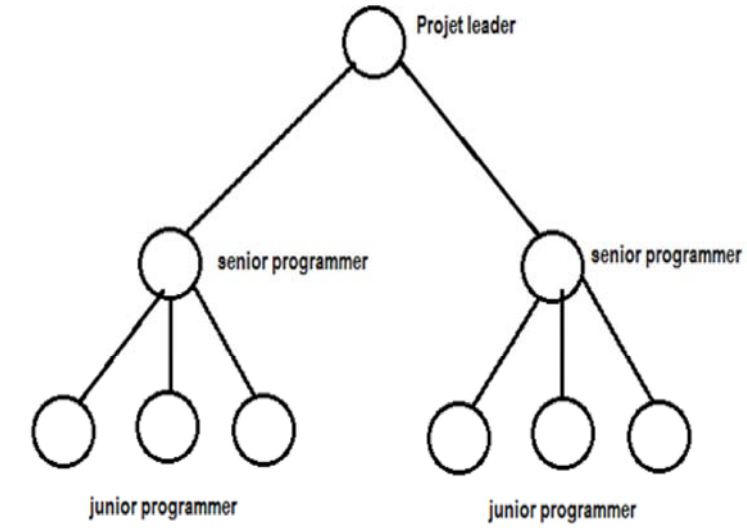


Figure 4.3 : Controlled decentralized Team

The junior programmers can communicate with them which in turn communicate with the senior programmer of their group. It is well suited for large project that are reasonably clear-cut. It is not suitable for research type of project or very simple project

4.3.5 Characteristics of a Good Software Engineer

A good software engineer should be:

- be familiar with software engineering principles
- have good domain knowledge (technical knowledge of the project area)

- show high motivation
- passing good programming abilities
- have leadership quality
- be intelligent and smart in handling the project
- possess adequate knowledge of basics of computer science
- follow descipline

CHECK YOUR PROGRESS

Q1. Which team structure is suitable for small well understood project?

(a) Chief team structure

(b) Egoless team structure

(c) decentralized team structure

(d) All of the above.

Q2. Give the structure of a controlled decentralized team.

4.4 PROJECT SCHEDULING

Software project scheduling is an important project planning activity under the supervision of project manager. In order to schedule a project, a project manager need to do the following:

- Identify the necessary activities needed to complete the project
- Sub-divide each activity into different needed tasks.
- Decide the dependency among the different tasks.
- Establish the time durations for the identified tasks
- Represent the information in the form of Activity Network.
- Decide the starting and ending date of the different various tasks.
- Determine the *critical path.* i.e. path which represents the duration of the project.
- Assign the needed resources to the tasks

The first step in the project scheduling is the identification of the initial necessary task needed to complete the project. The project manager decomposes the identified tasks into smaller tasks systematically. This is represented by *work breakdownstructure* (WBS). The next step is to find out the dependencies of the tasks. The dependencies of the tasks are represented by using *activity networks representation.* The next step is to allocate the

resources to each activity. The resource allocation is done using Gantt chart. The last step is to develop the project evaluation and review technique (PERT) chart.

4.4.1 Work Breakdown Structure

The *Work breakdown structure (WBS)* is used to sub-divide the initially identified activities of the project into smaller sub-activities. The lowest level of sub-activities is known as the tasks in a WBS. WBS is a tree representation of the activities, sub-activities and tasks to be carried out in order to solve the problem. Each node in this tree is represented by a rectangle. For each of the decomposed sub activities has an arrow from its parent activity. The main project represents the root of the WBS. The work breakdown structure of a *Demo Activity is shown in the figure 4.4*.In this figure, the *Main Activity* is composed of *Activities A, B, C, D* and *E* respectively. The activity *B* is again decomposed into sub-activities B1 and B2. Similarly, the activity *C* is again decomposed into sub-activities C1 and C2. The WBS helps the project manager to monitor the overall project.

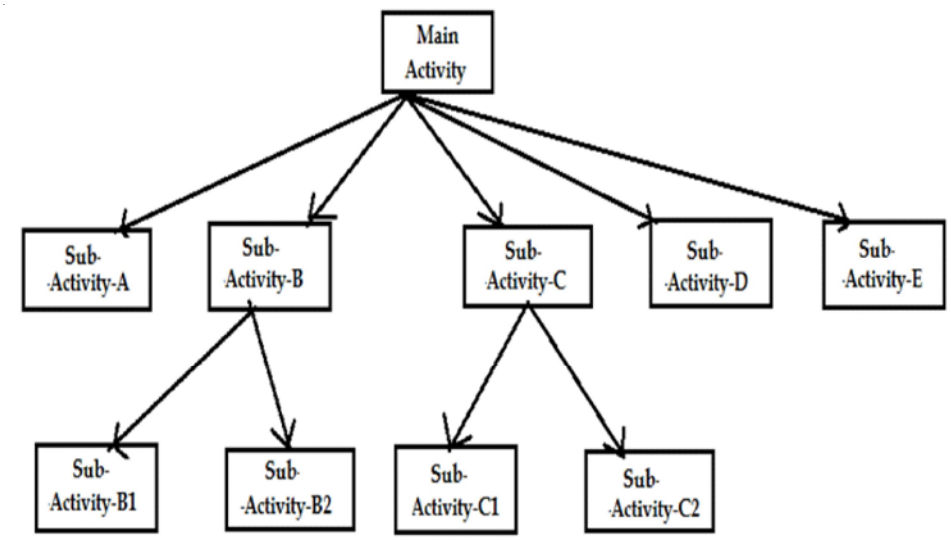


Figure 4.4: Work breakdown Structure of the *Demo Activity*

4.4.2 Activity Network Representation

An activity network representation shows the different activities involved in the projects, the interdependencies among them and the estimated durations for each of the activities in the proposed project. In the estimation of duration for the activities, the different project manager uses their experiences and past ideas. The *activity network representation of Demo-Activity project* is shown in the figure 4.5.

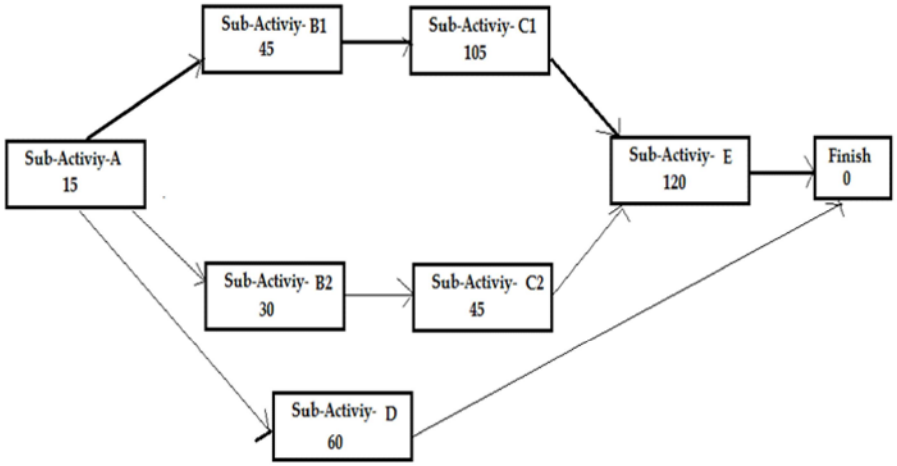


Figure 4.5: Activity Network Representation of *Demo Activity*

There are two variations in the activity network representations: *Activity of Node (AoN)* and *Activity of Edge (AoE).*

In the AoN, activities are represented by rectangular node and duration of the activity is shown alongside the node. The inter-dependencies are denoted by using directional edges. In the AoE, activities are represented by edges and duration of the activity is shown alongside each respective activity edge. Among this two, the AoN representation is most popular since it is easy to understand and revise.

The following project parameters are computed from the above activity networks of *Demo Activity* project:

Task Number	*Task Name*	*Duration (in days)*	*Dependent on Tasks*
T1	Activity-A	15	——-
T2	Sub-activity-B1	45	T1
T3	Sub-activity-B2	30	T1
T4	Sub-activity-D	60	T1
T5	Sub-activity-C1	105	T2
T6	Sub-activity-C2	45	T3
T7	Sub-activity-E	120	T5 and T6

4.4.3 Critical Path Method

Critical path method is an operational research technique to determine slack times for

tasks. A path in activity network representation is set of consecutive nodes and edges from start node to finish node.The *critical path* is the sequence of activities with the longest duration. A delay in any of these activities will result in a delay for the whole project. A critical task is one where slack time is zero. A critical path only contains the critical tasks from start node to finish node. In the computation of critical path, following parameters are needed to be calculated:

- **Minimum Time (MT)**: maximum of all the paths from start node to finish.
- **Earliest Start Time (EST)**: it is the time of a task which is the maximum of all the paths from start node to this task.
- **Latest Start Time (LST)**: difference between MT and maximum of all the paths from start node to finish.
- **Earliest Finish Time (EFT)**: the EFT of a task is the sum of EST and duration of the task.
- **Latest Start Time (LFT)**: the LFT of a task is the sum of LST and duration of the task.
- **Slack Time (ST)**: it is the total time that a task can be delayed. ST is calculated as the difference of LST and EST or LFT and EFT.

The calculated CPM parameters for the above *demo activity* project are given below:

MT=285

Task Name	**EST**	**EFT**	**LST**	**LFT**	**ST**
Activity-A	0	15	0	15	0
Sub-activity-B1	15	60	15	60	0
Sub-activity-B2	15	45	90	120	75
Sub-activity-C1	60	165	60	165	0
Sub-activity-C2	45	90	120	165	75
Sub-activity-E	160	285	165	285	0
Sub-activity-D	15	75	225	285	210

So, the critical path is consisting of *Activity-A*, *Sub-activity-B1*, *Sub-activity-C1* and *Sub-activity-E* represented by the dark edge in the figure 4.5.

4.4.4 Pert Chart

PERT (Project Evaluation and Review Technique) is a representation of Activity Network where activities are represented by boxes and arrows represents their dependencies. In

a PERT chart*pessimistic*, *likely*, and *optimistic estimates* are madefor each task instead of making a single estimate. This makes critical path analysis in PERT charts very complex. The PERT chart representation of the above *demo activity project* of figure 4.5 is shown in figure 4.6. PERT charts are a more complicated form of activity chart. With the use of PERT chart monitoring the timely progress of activities is possible.

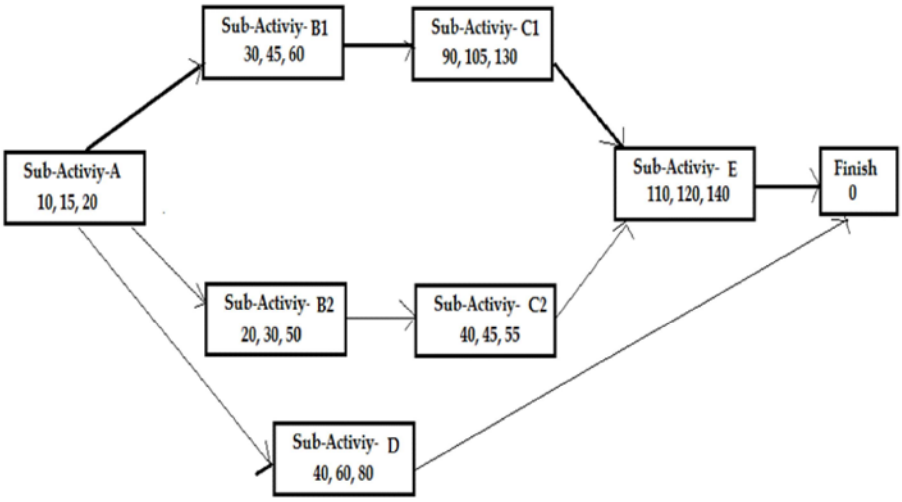


Figure 4.6: PERT Chart of *Demo Activity* in fig. 4.5.

4.4.5 Gantt Chart

A Gantt chart is a type of bar chart. It was first developed by Henry Gantt in the 1910s.It is generally used to allocate resources to activities. The resources may include staff, hardware, and software etc. In a Gantt, each bar represents an activity. The bars are drawn along a time line. Each bar consists of a white part and a shaded part. The shaded part of the bar shows the time estimated to take by the task. The white part shows the *slack time*, that is, the latest time by which a task must be finished. Gantt chart representation of a project schedule is helpful in planning the utilization of resources. A Gantt chart representation for the *demo activity* project problem of fig. 4.5 is shown in the figure 4.7.

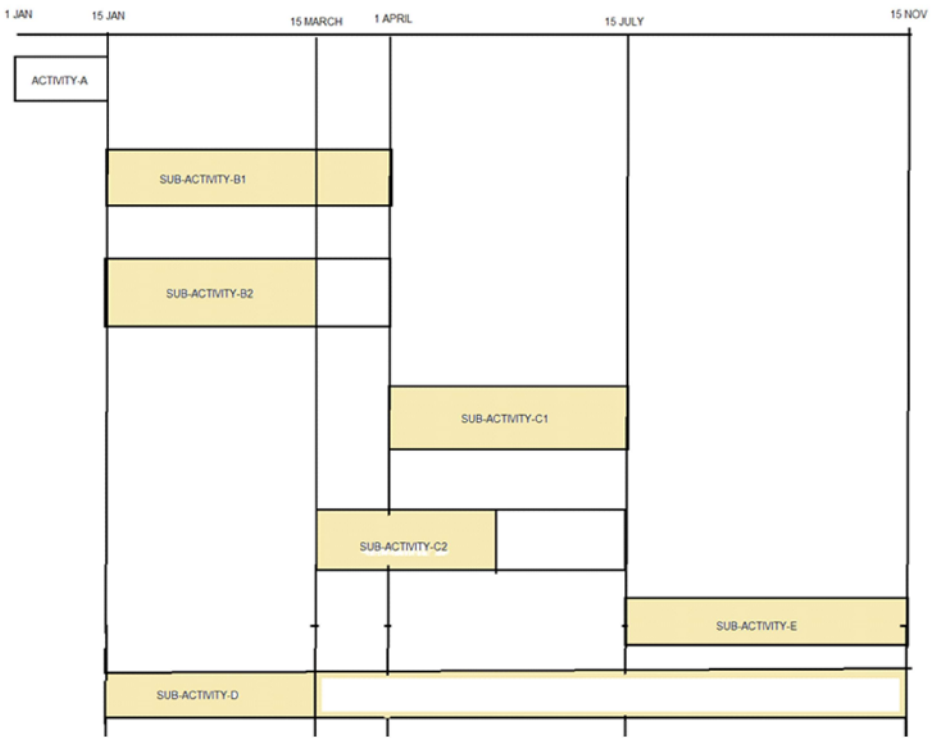


Figure 4.7: Gantt Chart of *demo-activity project*

CHECK YOUR PROGRESS

Q3. During project scheduling, resource allocation to different activities is done using which of the following representations:

a) PERT Chart b) Gantt Chart

c) Work breakdown structure

d) Acclivity Network Representation

Q4. Which of the following estimation is carried out first by a project manager during project planning?

a) estimation of cost b) estimation of the duration of the project

c) project size estimation d) estimation of development effort

Q5. Putnam's model is a _____.

a) Expert judgement model b) Delphi Cost estimation model

c) Regression Model d) Algorithmic estimation model.

Q6. The technology constant for a poor environmental project is ____.

a) 2 b) 4 c) 8 d) 11

4.5 LET US SUM UP

- Project Scheduling includes Project size, effort and duration estimation.
- LOC and FP are two metrics used for software cost estimation.
- COCOMO model has three variations such as *basic, intermediate* and *complete.*
- Software staffing are three types: *Chief team structure*, *Egoless team* and *Mixed controlled* team structure.
- Work breakdown structure is helpful to understand the activities and their interdependencies
- Activity network and Gantt chart is helps the project manager in monitoring and timely management of the project
- Gantt chart is used to allocate and maintain different resources available during the development of the project.

4.6 FURTHER READINGS

- Mall, R. (2018). Fundamentals of software engineering. PHI Learning Pvt. Ltd..
- Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.
- Kumar A., Kumar A. Software Engineering. Vayu Education of India

4.7 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: (a)

Answer to Q2: Controlled decentralized team structure consists of a project leader, who has a group of senior programmer. Each of the senior programmers again has a group of junior programmer under his leadership. The project leader communicates with the senior programmers and the senior programmers communicate with the junior programmers.

Answer to Q3: (b)

Answer to Q4: (c)

Answer to Q5: (d)

Answer to Q6: (a)

4.8 MODEL QUESTIONS

Q1. What is project staffing?

Q2. Describe briefly the different team structures used in project staffing.

Q3. Differentiate between egoless team and Chief programmer team structure.

Q4. What is Work Breakdown diagram? What is the importance of work breakdown structure?

Q5. What is Activity Network representation? Differentiate between Activity of Node and Activity of Edge representation.

Q6. What is Critical path? What are the different parameters needed to be calculated for critical path?

Q7. What are PERT chart and Gantt chart? How is it different from PERT chart?

Q8. Consider a project with following tasks information

Task Number	*Task Name*	*Duration (in days)*	*Dependent on Tasks*
T1	Specification	10	-----
T2	Design GUI	30	T1
T3	Database module	30	T2
T4	Code module	90	T1
T5	Integration	90	T4
T6	Testing	105	T5
T7	Documentation	110	T3 and T6

a) Draw the activity network representation.

b) Find out the Critical path for the above project.

UNIT 5: ROLE OF SYSTEM ANALYST

UNIT STRUCTURE

5.1 Learning Objectives
5.2 Introduction
5.3 Systems Analysis and Design (SAD)
5.4 Responsibilities of Systems Analyst
5.5 Skill Set Required to Perform Systems Analysis
5.5.1 Analytical Skill
5.5.2 Interpersonal Skill
5.5.2.1 Communication Skill
5.5.2.2 Interview and Listening
5.5.2.3 Questionnaire
5.5.2.4 Written and Oral Presentation
5.5.2.5 Working Individually and Coordinating with a Team
5.5.2.6 Managing Expectation
5.5.3 Technical Skill
5.5.4 Management Skill
5.6 Role to be played by System Analyst
5.7 Let Us Sum Up
5.8 Further Readings
5.9 Answers to Check Your Progress
5.10 Model Questions

5.1 LEARNING OBJECTIVES

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

- describe the responsibilities of a system analyst
- describe the skills required for system analysis
- describe the role of a system analyst
- define the various requirements which are to be developed

5.2 INTRODUCTION

In the previous units, we have learned the concepts related to software project management, staffing and scheduling. Different project estimation, empirical estimation and heuristic techniques have been discussed in detail there. In addition to these, the units also covered COCOCMO model and risk management issues were also covered. Different cost estimation methods like expert judgment method, Delphi cost estimation

along with software cost estimation metrics like lines of code and function point metrics were the other topics besides the different project scheduling representations like work breakdown structure, PERT charts, GANTT charts which have been covered in detail in the previous units.

In this unit, we will discuss issues related to system analysis and design. Therefore, this unit will offer a detailed analysis of the skills needed for a system analyst and the responsibilities of a system analyst. In the next unit, we will discuss the issues related to requirement analysis and specification.

5.3 SYSTEM ANALYSIS AND DESIGN (SAD)

In business, system analysis and design refers to the process of examining a business situation with the intent of improving it through better procedures and methods. System analysis and design relates to shaping organizations, improving performance and achieving objectives for profitability and growth. The emphasis is on systems in action, the relationships among subsystems and their contribution to meeting a common goal. Looking at a system and determining how adequately it functions, the changes to be made and the quality of the output are the essential parts of system analysis. Organizations are complex systems that consist of interrelated and interlocking subsystems. Changes in one part of the system have both anticipated and unanticipated consequences for other parts of the system. The systems approval is a way of thinking about the analysis and design of computer based applications. It provides a framework for visualizing the organizational and environmental factors that operate on a system. Among the positive consequences are the improved performance and a feeling of achievement with quality information. Among the unanticipated consequences the possible threat to an employee's job, a decreased morale of the personnel due to lack of involvement and a feeling of intimidation by the users due to computer illiteracy are common. The analyst's role is to remove such fears and make the system a success. System analysis and design focus on systems, processes and technology.

System development can generally be thought of as having two major components- *system analysis* and *systems design*. System design is the process of creating a new business system or one to replace or complement an existing system. But it is essential to thoroughly understand the old system throughly and determine how information technology can best be employed to make its operations more effective, prior to designing a new system. Thus, system analysis is the process of collecting and interpreting facts, identifying problem, and using information to suggest improvements to the existing system. This is the job of a System Analyst.

5.4 RESPONSIBILITIES OF SYSTEMS ANALYST

System analysis is the process of gathering and interpreting the facts, using the existing information to improve the system. A system analyst is a designated person who is responsible to study an existing system and design a new system. A system analyst has many roles to play such as an investigator, planner, designer, modulator, communicator, implementer, trainer, change agent, architect, psychologist, sales person, motivator, politician etc.

The following are the specific responsibilities of a system analyst:

- Analyze the existing system and make a requirement list by discussing with the users.
- Prepare a conceptual (logical) design for the system based on the requirements.
- Establish the boundaries of the system to use the inputs, outputs and interface.
- Define the functions to be performed and the parameters to measure performance.
- Find out the internal structures of the system and their dependencies.
- Prepare mathematical models to support the evaluation of system performance.
- Make alternative solutions and their weighted evaluation to choose the best.
- Decompose the system into various logical sub-systems to be integrated later.
- Participate in system development, testing, integration, and implementation.
- Associate with users, developers, and management for steering the project.
- Act as a leader in all the phases during the system development.
- Work as a change agent and catalyst for process development.
- Prepare the project plan and schedule for phase wise project completion.
- Determine the system reliability, availability, and quality.
- Prepare system development cost estimates and perform cost-benefit analysis.

5.5 SKILL SET REQUIRED TO PERFORM SYSTEMS ANALYSIS

Fundamentally for system analysis a detailed understanding of all important aspects of the operational area are required. For this reason, analysts interact with the employees and the managers of the client on a continuous basis to develop an in-depth understanding of the business processes and to seek answers of the following questions.

- What operations are being performed at present?
- How are the tasks being performed?
- How frequently does the task take place?
- What is the volume of transactions?
- How well is the task being performed?
- Does a problem exists?

- What are the causes behind the problem?

Once the processes are fully understood, the analyst identifies the key features of the proposed system.

An analyst must have several skills to effectively carry out the job. In order to become a successful systems analysts, one should have the four skills i.e. : a) *analytical* skills, b) *interpersonal* skills, c) *technical* skills and d) management skills. Analytical skills include the ability of systems thinking, which provides a foundation of the proposed system. Interpersonal skills focus on the interaction of the analysts with the people in the organization and building positive relationships. Technical skill requires the knowledge after present along technologies with emerging information technologies. Management skills are very useful for anyone in a leadership role. The analyst is expected to use the organizational resources in the most productive manner as possible where managerial skills are required.

5.5.1 Analytical Skills

The concept of system is central to system analysis and design. As we already know, a system is an interrelated set of components with an identifiable boundary, working together for some specific goals. System analyst must understand the functioning of the organization thoroughly. Given below is a list of specific areas of an organization that an analyst should understand.

How tasks are actually performed in an organisation?

- Policies
- Standards and Procedures
- Formal organisation structure
- Job description

Familiarisation with the organisation's culture

- Level of control exercised by key personnel
- Expert in different subjects areas
- Crucial incidents in the history of the organization
- Informal organization structure
- Values and mission

Other Factors :

- Government policies
- Products, services and markets
- Competitors strategies
- Role of technology

Problem identification is another area that needs to be addressed carefully. In order to identify problems, the analyst must be able to visualize and compare the present

situation in an organization to the desired situation. An analyst must have the ability to perceive problems from a broader perspective.

After the problem is identified, the analyst must analyse the problem and determine how to solve it. During this phase, the analyst may come out with several alternatives after interacting with the users and the management, and by referring to organizational files and documents. This process might get prolonged but finally the best alternative is chosen. As soon as the analyst, the users and the management agree on a solution, a plan is devised for implementing it.

The analyst must be perceptive, but must not jump to a quick conclusion; he/she must be persistent to overcome difficulties and maintain a planned course of action in spite of setbacks.

5.5.2 Interpersonal Skill

The job of a system analyst requires extensive interaction with various types of persons or users. This is the reason why interpersonal skills are considered as most important for a systems analyst.

The various interpersonal skills essential for system analysis are: communication skills; ability to work effectively with a team thus facilitating group tasks; and managing expectations of the users and the managers.

5.5.2.1 Communication Skill

Communication is the process of exchanging feeling with others. It is a bridge between the user and the system analyst. It involves a systematic and continuous process of telling, listening and understanding.

There are many different methods of communication such as: written (letters, reports, memos); verbal (face-to-face discussion, interviews, meetings); visual (charts, diagrams, presentation slides) and non-verbal. The analyst must have proficiency on as many forms of communication as possible.

5.5.2.2 Interview and Listening

Interview is the most common method used in information gathering for system analysis. Initially an analyst might have to spend a lot of time in interviewing the users about their work. Merely asking question does not serve solve the purpose. The analyst must intently listen to the answers. Careful listening helps an analyst develop good rapport with the user and understand the problem in all its perspectives.

5.5.2.3 Questionnaire

Despite being one of the effective techniques for interacting with people, interviews can prove to be very expensive and time-consuming. Questionnaires, on the other hand, are less effective than interviews because they do not provide direct means to ask follow-up questions. However, there are many advantages of using questionnaires for gathering information. They are comparatively less expensive.

5.5.2.4 Written and oral presentations

The progress of the project needs to be documented at various stages of the systems development life cycle. This documentation is essential not only to track the activities of the development process but also to serve the personnel who are given the responsibility of maintaining the system later. At each stage of the systems development life cycle, the analyst is required to prepare and file a report. For an analyst to be effective, he/she should be able to write the reports clearly and persuasively. The general principles of presenting information, so that it is comprehensible are: directness, conciseness, organization and clarity. Some projects may also require the analyst to orally present the information.

5.5.2.5 Working Individually and Coordinating with a Team

An analyst is often required to work alone in some stage of a system development project. In such cases, he must be able to keep up to the schedule, communications and deadlines. He/she will always have to work with the team for achieving project goals. Effective teams are built through a substantial commitment of time, energy and goodwill on the part of all the members.

5.5.2.6 Managing Expectations

Managing expectations start with the preparation of the business case for the system and go all the way through training people to usually the proposed system. The prospective users of the system might have certain ideas and may be apprehensive about the working of the proposed system. They might also have certain expectations from the system.

5.5.3 Technical Skill

The analyst should continually upgrade themselves about himself/herself recent developments in the field of information technology, techniques and methodologies. As information technology, techniques and methodologies keep on changing at a fast rate, it becomes essential for the system analyst to keep abreast of the latest developments. The analyst must have a knack for representing, or modeling, various aspects of information

systems. These skills help an analyst not only to perform the assigned task dexterously, but also to interact with other systems development personnel.

To stay up-to-date and versatile, the analyst must:

- Read books and trade journals
- Take membership of professional societies and actively take part in meetings.
- Attend training sessions of courses offered by the organization.
- Regularly browse web sites that offer the latest information about the industry.
- Take part in electronic boards, newsgroups or conferences, or local, national or international networks.

The analyst should make every possible effort to get the latest status about the following hardware and software:

- Programming languages
- Operating Systems
- Database and file management systems
- Data communication equipments and software
- Systems development tools and environments
- Web development tools and languages
- Division support system generators and data analysis tools.
- Data security

Familiarization with modern methods and techniques for describing, modeling and building systems, also increases the efficiency of the analyst.

5.5.4 Management Skill

System analyst must possess the management skills as they assume the role of a team leader in almost all projects. An analyst must be adept in employing the organizational resources in the most in as productive manner as possible. Some of the management skills that an analyst should necessarily possess are:

- Resource management
- Project management
- Risk management
- Change management

Resource Management: The analyst must have the innate quality of optimizing the resources tactfully

- Anticipating resource usage (estimating requirements).
- Tracking and accounting for resource utilization.
- Mastering how to use resources efficiently.
- Assessing the quality of resources used.
- Ensure proper resource utilization
- Releasing resources when no longer required.

Project Management: The prime objective of project management is to ensure that projects are completed successfully within specified budget and schedule. It aims at helping managers to keep track of the progress. An analyst should start breaking a project into several independent and manageable tasks. They should then delegate the responsibility of each of these tasks to people based on their skill sets. The analyst uses various tools and techniques to manage projects. For example, Gantt and PERT charts are powerful graphical techniques used in planning and controlling projects.

Risk Management: Risk Management is the ability to foresee what might go wrong in a project and take appropriate preventive and corrective action. After identifying the risks, the analysts must make every possible endeavor to reduce them. Risk management also includes knowing where to employ resources and prioritizing tasks to attain the maximum benefit.

Change Management: Bringing in a new or improved information system into an organization is a change process. It brings with it lot of challenges in the form of resistance from the prospective users. Therefore, change management is a very important skill for system analyst.

System analysts are required to ensure smooth transition to a new and improved information system. Change management also deals with the ability to deal with technical issues to change by involving them in the design and implementation stages of the project. Extensive interaction with the users makes this challenging.

5.6 ROLES TO BE PLAYED BY SYSTEMS ANALYST

The major responsibilities of system analyst are to identify the information needs of an organization and design an information system that caters to these needs. People who are involved in developing information system for organization can be grouped into the following three categories: managers, users of the system, and designers and programmers who implement the systems. The system analyst organises, the endeavours of these groups to effectively design and implement computer-based information system.

The system analyst takes on various roles to carry out their tasks effectively. Some of the major roles that the analysts have to take are discussed below:

- **Change Agent:** It is the system analyst who, after identifying the problems

in the existing system, suggests changes. Making the users to accept change, i.e. the proposed system, with a minimum of resistance involves careful planning. The analyst makes the users ready for the change by involving them in the design and implementation stages of the project. Extensive interaction with the users makes this challenging task less daunting.

- **Overseer:** The analyst highlights the inefficiencies of the existing system by collecting all possible information about the same. They not only suggest changes to solve the problems in the existing system, but also ensure timely completion of the project, taking the cost and quality into consideration.
- **Designer:** The task of the designer or architect is to give concrete shape to the clients, abstract design requirements. An analyst creates a detailed physical design of the candidate system.
- **Psychologist:** In system development, systems are built around people. The analyst plays the role of a psychologist in the way he/she reaches out to people, interprets their thoughts, and assesses their behavior, and draws and make conclusion from these interactions. Understanding inter-functional relationships is crucial. It is also important that the analyst be aware of people's feeling and be prepared to handle issues in a graceful manner. The art of listening is important in evaluating responses and feedbacks.

Salesperson: The analyst starts by selling the idea of change and continuously works towards making the candidate system acceptable to the prospective users. Hence sales skills and persuasiveness are important qualities of system analysts. System acceptance is attained through user participation in its development, effective user training and proper motivation to use the system. The analyst's role as a motivator becomes evident during the first few weeks after implementation and during the time when the users are trained to work with the proposed system.

CHECK YOUR PROGRESS

Q1. The prime objective of project management is _____.

a) To ensure smooth transition to a new and improved information system.

b) To employ the organizational resources in the most productive ways

c) To ensure that projects are not delayed and do not go over budget

d) None of the above

Q2. Microsoft project for windows is a _____.

a) Risk management system b) Project manage-ment system c) Change management system

d) None of the above

Q3. Which of the following is not a skill requirement for the job of a system analyst?

a) Flexibility b) Knowledge of people

c) Ability to communicate d) None of the above

Q4. For which of the following phases, systems analyst is not responsible?

a) Maintenance phase b) Design phase

c) Testing phase d) Operation and use

Q5. What are the characteristics which are required in a systems analyst so as to understand the user requirements?

a) Knowledge of business b) Communication skills

c) Both of the above d) None of the above

Q6. State whether true or false:

i. It is not mandatory for an analyst to understand the management structure and the relationship between the departments in an organization.

ii. The progress of the project needs to be documented at various stages of the systems development process.

iii. Gantt and PERT charts are powerful graphical techniques used in system development life cycle.

iv. Project management is the ability to foresee what might go wrong in a project.

v. Successfully managing user expectations is related to successful systems implementation.

5.7 LET US SUM UP

- System analysis is the process of understanding a business system and its processes.
- System design is the process of architecting a new business system or one to replace or complement an existing system.
- The person who carries out the process of systems analysis and design is called system analyst.
- A system analyst needs to have interpersonal, technical and management skills.
- A system analyst may be responsible for system analysis, design and programming depending on the organization size. The title given to the system analyst also depends on the organization type or responsibilities given to him/her.
- With the advent of easier computing and friendly development tools, changes are coming. End user computing is becoming popular.
- System analysts play a variety of roles like that of a change agent, overseer,

designer, psychologist and salesperson.

5.8 FURTHER READINGS

- Sommerville, I. (2013). *Software Engineering: Pearson New International Edition.* Pearson Education Limited.
- Mall, R. (2014). *Fundamentals of software engineering.* PHI Learning Pvt. Ltd.

5.9 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: i. b

Answer to Q2: ii. b

Answer to Q3: iii. d

Answer to Q4: iv. b

Answer to Q5: v. c

Answer to Q6: i. False

ii. True

iii. True

iv. True

v. True

5.10 MODEL QUESTIONS

Q1. What is system analysis and design?

Q2. What kinds of skills are required for a system analyst?

Q3. How does a system analyst bring about “change” in a business system?

Q4. Discuss the various responsibilities of a system analyst.

Q5. “A system analyst also has a role of psychologist”. Discuss.

Q6. How does the size of organization affect the responsibilities of the system analyst?

Q7. What are the various skills required of an analyst in the various stages of the system development life cycle?

Q8. System analysis and system design are two different activities. What is the difference between them?

UNIT 6: REQUIREMENT ANALYSIS AND SPECIFICATION

UNIT STRUCTURE

6.1 Learning Objectives
6.2 Introduction
6.3 Fact Gathering
6.4 Requirement Analysis
    6.4.1 User Transaction Requirements
    6.4.2 User Design Requirements
6.5 Feasibility Study
6.6 System Requirement Specification (SRS)
6.7 Functional Requirements of SRS
6.8 Organisation of SRS
6.9 Let us Sum Up
6.10 Further Readings
6.11 Answers to Check Your Progress
6.12 Model Questions

6.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- define and describe the term 'Fact Gathering'
- distinguish between the different fact finding techniques
- analyse the requirement of system analysis procedure
- study the viability to develop a system
- describe the requirement specification of system and its functional requirements
- explain the organisation of SRS

6.2 INTRODUCTION

By now you must have already been familiar with the System and their types. We have already discussed in detail the software engineering approaches, their needs, qualities and various characteristics. To have a good system and a good software, there is a need for detail investigation and analysis of the existing system. We should be clear about the requirement of the proposed system. To implement a system it is necessary to know for the developer about the standard organisation of the system. All these concepts

are explained in this unit. This unit begins with the definition of system analysis and their requirements in the development process. We will discuss about system requirement specification (SRS) to know about what the proposed system will do without knowing how it will do. The standard organisation of SRS, its requirement in system will also be discussed in this unit besides the importance of system analysis, feasibility study and requirement specification. In the next unit, we will explore the concepts related to software design like function oriented design and object oriented design in addition to topics like cohesion and coupling.

6.3 FACT GATHERING

We know that data collection is an important part of system analysis. This is needed for geting some new ideas from the gathered data after analysis. This is basically done by the system analyst. To gather information from users an analyst should use some fact finding techniques which are discussed below:

Review of written documents:

Many kinds of records and reports provide valuable information about organisations and operations. They include written policy manuals, regulations, standard operation procedures used by the organisation as a guide. These do not show actual activities, but will help in understanding the system. While observing the current reports one should scrutinise the data present on the report. The analyst should find out from the existing report how useful they are to the user, how they are filled out, what changes need to be made, and how easy they are to read.

On-site observations:

This method helps to obtain first hand information on how activities are being carried out. It is one of the most difficult fact-finding techniques and an analyst follows a set of rules as an observer. On-site observation is useful when the analyst needs to observe how processes are carried out and whether specified steps are actually followed. The analyst observes the physical layout of the existing system, the location and movements of people, and the work flow.

If on-site observation is to be done properly in a complex situation, it can be time consuming. Proper sampling procedures must be used to ascertain the stability of the behaviour being observed.

Interview method:

As discussed in the previous unit, analyst uses interview for collecting information from individuals or groups who are generally current users of the existing system or potential users of the proposed system. They may be managers or employees who provide data for the proposed system or who will be affected by it.

Interview is a time consuming method but is the best source of qualitative information. It allows an analyst to discover areas of misunderstanding, unrealistic expectations and indication of resistance to the proposed system.

Questionnaire:

The use of questionnaires allows collection of data from large number of persons. In this method the analyst cannot observe reactions, motivation or expressions of the respondents. The quality of the response is judged in terms of its reliability and validity. **Reliability** means that the information gathered is dependable enough to be used for making decisions about the system being studied. **Validity** means that the questions asked are worked to elicit the intended information. So, the reliability and validity of the data gathered depend on the design of the questionnaire.

Questionnaires are of two types: Open-ended or Closed-ended. **Open ended questionnaires** are used to learn feelings, opinions, general experiences on process detail or problem. A **closed questionnaire** offers specific responses which have to be selected.

CHECK YOUR PROGRESS

Q1. Fill in the blanks:

a) Data collection is basically done by _____.

b) Gathering information for analysis is known as _____.

c) The first-hand information of fact gathering is the _____.

d) Interview method can be done _____ or _____.

e) The quality of the questionnaire is judged from its _____ and _____.

6.4 REQUIREMENT ANALYSIS

After preliminary investigation of the proposed system the next step is the software requirement analysis. For a large software system, Requirement Analysis is the most difficult and intractable activity. It is very error-prone. Requirement Analysis is the heart of the system development life cycle. A requirement is a feature that must be in-

cluded in a new system and it needs detailed understanding of all the important facets of the system. It may include a way of capturing or processing data, producing information, controlling a business activity, or supporting management. It is the process of discovery or evaluation. According to the IEEE standard, Requirement can be defined as:

1) A condition of capability needed by a user to solve a problem or achieve an objective
2) A condition or a capability that must be met or processed by a system to satisfy a contract, standard, specification or other formally imposed document.

When a system grows more complex and the goal of a system cannot be easily realised, then the need of requirement analysis arises. Now, for a large software system it is more difficult and more error-prone. Hence, requirement necessarily involves what some people have in their mind.

It is important to know that the software requirements we are dealing with the requirements of the proposed system and it is the last step in the planning phase of the software life cycle. It satisfies the following objectives:

i) By uncovering the flow and structure of information, it provides a foundation for software development.
ii) By identifying interface details, providing an in-depth description of functions, determining design constraints and defining software validation requirements, it describes the software.
iii) Establish and maintain communication with the user and the developer to satisfy the above two objectives.

To achieve the above objectives the requirement analysis has to encompass the different tasks. The whole analysis procedure we can divide into four stages. These we :

i) Problem recognition: The aim of the analyst is to recognise the basic problem of the system as perceived by the user. In this stage the first work is to study the system specification and plan the software according to system context to prepare planning estimate. In this period the analyst must establish contact with management and technical staff of the user and the software developer.

ii) Evaluation and synthesis: This is also one main area of analysis procedure. During this section the analyst evaluates the flow and structure of information, define all software function in detail, establish system interface characteristics, and uncover design constraints.

iii) Specification: A formal requirements specification and a preliminary User's Manual are documented to define the characteristics and attributes of the software.

iv) **Review**: The user manual prepared in the previous stage encourages the user to review the software from a human-engineering perspective. Review is very much essen-

tial in improving and modifying the system according to the user's requirements. In this stage the cost and schedule estimates are also reviewed.

6.4.1 User Transaction Requirements

User requirements specify the acceptable level of user performance and satisfaction with the system. The User Transaction Requirements typically describe the needs, goals, and tasks of the user. It usually not reflects the actual person who will be using the software and projects are often tailored to the needs of the project requestor, not to that of the *end*-user of the software. Any user requirements document defines and describes the end-user, and any measurements of quality or success are taken with respect to that end-user. User requirements are usually defined after the completion of **task analysis**, the examination of the tasks and goals of the end-user.

6.4.2 User Design Requirements

A critical part of any software development project is the requirements gathering. Good user requirements lead to better estimates, improved customer satisfaction, reduced cost and a shorter duration. The "build to" "code to" and "buy to" requirements for products and "how to execute" requirements for processes expressed in technical data packages and technical manuals through different design specification. This expression is must to understand the system better for the user. The design specification addresses the look and feel of the interface with rules for the display of global and particular elements.

An integral part of an effective user-centred design (UCD) process is the gathering of requirements and the identification of common user tasks. This part of the UCD process is just as important in software design.

The design process should take care of the following:

- Identification of reports and outputs the new system should produce.
- Scrutinize the data present on each report/output.
- Sketch the input form and output is expected to appear at the completion of the system.
- Description of data to be input, calculated or stored.
- Individual data items and calculation procedures are written in details.
- The procedures written should tell how to process the data and produce the output.

The document produced at the end of this activity is called the "**Design Specification**". This document should have charts, tables and special symbols to portray the design.

CHECK YOUR PROGRESS

Q2. Fill in the blanks:

a) Software requirement analysis step is next to _____ step

b) Requirement analysis provides a foundation for _____.

c) Requirement analysis maintains communication with the _____ and the _____.

d) In the _____ period the analyst must establish contact with management and technical staff of the user and the software developer.

e) In the _____ section, the analyst establishes system interface characteristics.

f) User manual are prepared in the _____ stage of requirement analysis process.

g) The _____ typically describe the needs, goals, and tasks of the user.

h) _____ is one of the important steps in software design.

i) The document produced at the end of this activity of requirement analysis is the _____.

Q3. Write True or False:

a) Software requirement is the analysis of the existing system.

b) Analysis specifies how a system will do.

c) An analyst must be a good communicator.

d) For large software system, requirement analysis is the most difficult and intractable activity.

e) The design process should take care of the scruting of the data present on each report/output.

j) Individual data items and calculation procedures are written in detail in _____.

6.5 FEASIBILITY STUDY AND ITS TYPE

It is the preliminary investigation and the second phase of system development life cycle. This starts as soon as someone, either a user or a member of a particular department, recognises a problem or initiates a request, to modify the existing system. An important outcome of this study is to determine whether the system requested is possible or not. It is the test of a system proposal according to its workability, impact on the organisation, ability to meet user needs and effective use of resources.

The objective of feasibility study is not to solve the problem but to acquire a sense of its scope. The major purposes of this study are listed below:

a) Identify the responsible users and develop an initial "scope" of the sys-

tem. This may involve conducting a series of interviews to see which user(s) are involved (or affected by) the proposed project and which are not. The scope of the system is determined based on the information collected from the users.

b) **Identify current deficiencies in the user's environment.** This usually consists of a simple narrative list of functions that are missing or operating unacceptably in the current system. It also gives the details of modifications necessary to the existing system.

c) **Determine objectives for the new system.** It is also a narrative list consisting of existing functions that need to be re-implemented and added.

d) **Determine whether it is feasible to automate the system**. Accordingly it suggests some acceptable solutions. This involves very crude and approximate estimates of the schedule and cost to build a new system.

The result of the feasibility study is a formal report that consists of the following:

1. Statement of the problem.
2. Summary of findings and recommendations.
3. Conclusions.

There are three major areas of feasibility study. They are:

a) **Technical feasibility:** The analyst must find out whether current technical resources in terms of hardware and software are available in the organisation or not and is the organisation and to what extent it can support the proposed addition.

b) **Economic feasibility:** Economic and financial feasibility analysis is the most frequently used method for evaluating the effectiveness of a candidate system. The basic resources are:

- Management time
- Time spent by the systems analysis team
- Cost of doing the full systems study
- Estimated cost of hardware and software.

c) **Operational feasibility:** Once it is determined that the system is both technically and economically feasible then it has to see the operational feasibility. It is dependent on human resources for the project. It refers to projecting whether the system will operate and be used once it is installed. In this step an estimate should be made how strong a reaction the user staff is likely to have toward the development of a proposed system.

The document to be produced at the end of this activity is called **Feasibility Study Report.**

CHECK YOUR PROGRESS

Q4. Fill in the blanks

a) The first phase of the system development life cycle is _____.

b) Feasibility Study is the _____ study of the system development life cycle.

c) The objective of feasibility study is not to solve the problem but to acquire a sense of its _____.

d) Feasibility study involves very crude and approximate estimates of the schedule and cost to build a _____.

e) Economic and financial feasibility is one important part of the _____.

6.6 SYSTEM REQUIREMENT SPECIFICATION (SRS)

Specification is the document developed during the analysis phase. It is the main part of software development process because the success of a software development entirely depends on it. The goal of the requirements activity is to produce SRS that describes *what* the proposed system should do without describing *how* the system will do. This document extends scope by establishing a complete information description, a detailed functional description, appropriate validation criteria, and other data pertinent to requirement. It is an agreement between the user and the developer.

The *introduction* section states the goals and objectives of the software. The *information description* provides a detailed description of the problem that the software must solve. Both information flow and structure are documented. The *functional detail* describes the procedures and functions required in the system. The *validation criteria section* describes the implicit review of information and functional requirements. Finally the SRS includes other planning phase documentation, technical references vendor literature and standards. Tabular data, detailed description of algorithms, charts, graphs and other material are presented as appendices.

The advantages of using SRS are:

a) It is the bridge between client and developer through which the client and user needs are accurately specified to the developer. SRS establishes the basis for agreement between the client and the supplier on what the software product will do.

b) An SRS provides a reference for validation of the final product. Without a proper

SRS there is no way a client can determine if the software being delivered is what was ordered, and there is no way the developer can convince the client that all the requirements have been fulfilled.

c) A high quality SRS is a prerequisite to high quality software.

d) An SRS reduces the development cost.

6.7 FUNCTIONAL REQUIREMENTS OF SRS

Functional requirement of SRS describes the expectations of the system, i.e. what the system is expected to do. It is needed to explain the process that the system will carry out. It includes details of input, its source, ranges, all operations on input data and expected outputs and also their relationships. SRS is needed basically to make a proper understanding between the clients and the software developer.

An important part of SRS is in the abnormal situation of the system that occurs due to invalid inputs or due to some other errors. The functional requirement of SRS should clearly mention the system's performance in this type of situation and how it will come back to the normal situation etc. The special conditions should always be mentioned in the functional SRS. For example, an airline will never book ticket even for a valid passenger if the airplane is fully booked.

6.8 ORGANISATION OF SRS

It is necessary to have a specific structure for all the requirements of the system. There are many ways to organise the requirements document, but it should be easy to understand for others. As per the IEEE standard, the SRS is organised with various subsections show below:

1. Introduction
    - 1.1. Purpose
    - 1.2. Scope
    - 1.3. Definitions
    - 1.4. References
    - 1.5. System Overview
2. Overall Description
    - 2.1. Product Perspective
    - 2.1.1. System Interfaces
    - 2.1.2. User Interfaces
    - 2.1.3. Hardware Interfaces
    - 2.1.4. Software Interfaces
    - 2.1.5. Communication Interfaces

2.1.6. Memory Constraints
2.2. Product Functions
2.3. User characteristics
2.4. General Constraints
2.5. Assumptions and Dependencies
3. Specific requirements
3.1. External interface requirements
3.2. Functional requirements
3.3. Performance requirements
3.4. Logical database requirement
3.5. Software System attributes
3.5.1. Reliability
3.5.2. Availability
3.5.3. Security
3.5.4. Maintainability
3.5.5. Portability
3.6. Others.

Section 1 contains the purpose, scope, definition, references cited in the document and overview of the system. *Section 2* describes the general factors that affect the product and its requirements. *Section 3* describes the interface requirement that includes user interface, hardware interface and software interface of the system, functional requirement, external and database requirement in the system. It also describes the attributes used for the software. This section should describe all the details that the software developer need to know to design and develop the system.

In the functional requirement section, functional capabilities for all the modes of operations of the software for the system are specified. It is not describing the details about the algorithm, but should describe the link between the used input and desired output. For outputs, the destination of outputs, unit of measure, range of valid outputs, error messages etc. are to be specified.

CHECK YOUR PROGRESS

Q5. Write True or False:

a) SRS is the bridge between client and developer.
b) Interview is a time consuming method of data collection.
c) The detailed document after analysis of a system is the design specification.

d) A requirement is a feature that must be included in a new system.

e) Determining objective is work of fact finding stage.

6.9 LET US SUM UP

- Requirement Analysis is the heart of the system development life cycle. A requirement is a feature that must be included in a new system and it needs detailed understanding of all the important facets of the system.
- To evaluate the existing system and procedures and to analyse the candidate system with cost estimates are to be taken before designing a system. This is the preliminary investigation we should take care of. It is called feasibility study.
- An important outcome of feasibility study is that it tells us whether the requested system is feasible or not. After feasibility study a formal acceptance of the proposal is taken from the user. The called *feasibility report*.
- Determination of system requirement is the next step of feasibility study and for this detailed investigation is required. The success of a software development effort depends entirely on the complete specification of software requirements.
- Functional requirement of SRS describes the expectations of the system, i.e. what the system expected to do. It is needed to explain the process that the system will carry out.

6.10 FURTHER READINGS

- Jalote, P. (2012). *An integrated approach to software engineering.* Springer Science & Business Media.
- Mall, R. (2014). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd.
- Awad, E. M. (1985). *Systems analysis and design*. RD Irwin.

6.11 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: a) system analyst

b) fact gathering approach

c) On-site observations

d) individually & group wise

e) reliability & validity

Answer to Q2: a) preliminary investigation

b) software development

c) user & developer

d) Problem recognition

e) Evaluation and synthesis

f) Review

g) User transaction requirements

h) UCD process

i) Design Specification

j) UCD process

Answer to Q3: a) F

b) F

c) T

d) T

e) T

Answer to Q4: a) Feasibility Study

b) preliminary

c) scope

d) new system

e) resource determination

Answer to Q5: a) T

b) T

c) F

d) T

e) F

6.12 MODEL QUESTIONS

Q1. Discuss the objectives of Requirement Analysis to meet a successful system.

Q2. Differentiate between the terms Analysis and Specification.

Q3. What is feasibility study? Why is it important?

Q4. Discuss the different sections of SRS document. Why is SRS important?

Q5. What is design specification? What tools are used to represent a system in this period?

Q6. Discuss briefly the techniques used for collecting data in analysing a system.

Q7. How can you examine a good questionnaire? What are its types?

Q8. Discuss the standard organisational structure of SRS.

****

UNIT 7: SOFTWARE DESIGN

UNIT STRUCTURE

7.1 Learning Objectives
7.2 Introduction
7.3 Software Design
7.3 Qualities of a Good Design
7.4 Modules Concept
7.5 Characteristics of Modules
7.5.1 Cohesion
7.5.2 Coupling
7.6 Approaches to Software Design
7.6.1 Top Down Approach
7.6.2 Bottom Up Approach
7.7 Variants of Software Design
7.7.1 Function Oriented Design
7.7.2 Object Oriented Design (OOD)
7.8 Let Us Sum Up
7.9 Further Readings
7.10 Answers to Check Your Progress
7.11 Model Questions

7.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- define a good software design
- describe the role of smaller components (modules) in software design
- measure the dependability of the various modules with one another
- describe the two most basic approaches to software design – Top Down Approach and Bottom Up Approach
- describe the two variants of software design - Function oriented and Object oriented design.

7.2 INTRODUCTION

In the previous units, we have already discussed in detail the definition of system analysis and their requirements in the development process. In the previous unit, we have also discussed the importance of system analysis, feasibility study and requirement specifi-

cation. In addition to these topics, we have also discussed in system requirement specification (SRS) and its standard organisation structure.
In this unit our main concern is software design. Necessarily, we are discussing the qualities necessary for a good software design and the concept of modules along with cohesion and coupling are covered in this unit. Top down and bottom up approach to software design are covered in detail in this unit, besides the two variants of software design- Function oriented design and Object oriented design. In the next unit, we will explore the concepts related to function oriented design in detail.

7.3 SOFTWARE DESIGN

Software design is a series of action required to convert user requirement into some form which helps the developer in software coding and implementation. It is the first step in SDLC (Software Development Life Cycle), which is used to move the application from problem domain to solution domain. To assess the user requirements, an SRS (Software Requirement Specification) document is created. Software design tries to identify how to satisfy the requirements mentioned in SRS documents.

7.3.1 Qualities of a Good Design

The concept of a good software design can vary depending on the application being designed. Although there are many measures of software quality, the following are some major factors that define the quality of software.

1. Correctness: A good design should correctly implement all the functionalities identified in the SRS document.
2. Understandability: A good design should be easily understandable.
3. Efficiency: it should be efficient.
4. Performance: It should perform well in regular and stressful scenarios.
5. Interface: The user interface should be simple and elegant to use.
6. Maintenance: The developer must easily fix the bugs and add functionality without having to recreate the whole code.
7. Security and Safety: The software should not have bad effects on data / hardware. Proper measures should be taken to keep data secure from external threats.
8. Scalability: It should be very easy to be upgraded it for more works or for more number of users.
9. Modularity: Software must be made of modules which are independent to each other so that they can be modified and tested separately. These modules are than integrated to make the final software.

7.4 MODULES CONCEPT

A software system is divided into various discrete and separate modules, since smaller components are easier to maintain. These modules are required to do the tasks independently without overlapping with each other. The modular design follows the rules of divide & conquers problem solving strategy.

Some of the advantages of modularity are as follows:

- Concurrent execution can be made possible.
- Components with high cohesion can be required again.
- Program can be divided based on functional aspects.
- Smaller components are easier to maintain.
- Shorter learning time.
- Reduction in cost.

7.5 CHARACTERISTICS OF MODULES

A software system can be divided into modules based on some characteristics. These modules are then grouped and integrated to make the final software. The quality of these modules and interaction among them can be measured by cohesion and coupling.

7.5.1 COHESION

Cohesion is the intra dependability of the elements of a single module. In other words, it is the measure of the relative functional strength of a module. The higher the cohesion better is software design. There are seven (7) types of cohesions. They are discussed as follows.

1. **Coincidental Cohesion:** it is unplanned and random cohesion which might be due to the separation of program into smaller modules for the sake of modularization. Since it is unplanned it may create confusion to the programmers and is generally not accepted.
2. **Logical Cohesion**: when logically categorized elements are put together into a module, it is called logical cohesion.
3. **Temporal Cohesion**: When elements of a module are organized in such a way that they are processed at similar point of a time, then it is called temporal cohesion.
4. **Procedural Cohesion**: When elements of a module organized in such a way that they are executed sequentially then it is called procedural cohesion.

5. **Communication Cohesion**: When elements of a module execute sequentially and process the same data it is called communication cohesion.
6. **Sequential Cohesion:** when elements of a modules are grouped because the output of one element is input to another and so on, it is called sequential cohesion.
7. **Functional Cohesion**: it is considered to be the higher degree of cohesion and it is highly expected. The elements are grouped in such a way that they must contribute to single well defined function. The elements can then be reused.

7.5.2 COUPLING

Coupling is a measure of interconnection among modules in a software structure. It deals with interference and interaction between modules in a software system. If interference is low then better is the program. When, coupling is low then we say that software design is good. To be a good design there must be high cohesion and less coupling. There are five (5) levels of coupling. Ideally, no coupling is considered to be the best. They are discussed as follows.

1. **Content Coupling**: when a module can directly access or modify a content of another module, it is called content coupling.
2. **Common Coupling**: when various modules have read and write permission to some global data it is called common coupling.
3. **Control Coupling**: two modules are called control coupling if one of them decides to do the function of the other module or changes the flow of execution.
4. **Stamp Coupling**: when various modules share common data structure and work on different part of it, it is called stamp coupling.

Data Coupling: Data coupling is when two modules interact with each other by means of parameter passing

CHECK YOUR PROGRESS

Q1. _____ is the measure of degree of independence between modules.

a) Cohesion b) Coupling c) None of the mentioned

Q2. Which of the three comes under modularity principle?

a) Cohesion b) Coupling

c) Small module d) All of the mentioned

Q3. Which of the following is true?

a) Module coupling should be maximized

b) Module cohesion should be minimized.

c) Modules should not have access to the resources that are notrequired

d) Design with small modules is not good.

Q4. In the context of modular design which of the following combinations is desirable?

a) High cohesion, High coupling

b) High cohesion, low coupling

c) Low cohesion, high coupling

d) Low cohesion, low coupling

Q5. Match the following:

List 1	List 2
a) Data coupling	i. Module A and B have shared data.
b) Stamp coupling	ii. Communicates by passing data.
c) Common coupling	iii. Complete data structure is passed from one module to another.
d) Content coupling	iv. Control is passed from one module to another.

7.6 APPOACHES TO SOFTWARE DESIGN

Software design is a data flow based methodology. This approach begins with a system specification that identifies inputs and outputs and describes the functional aspects of the system. The system specifications are then used as a basis for graphic representation. There are two generic approaches for software design. They are:

a) Top down approach

b) Bottom up approach

7.6.1 TOP-DOWN APPROACH

We know that a system is composed of more than one sub-systems and it contains a number of components. Also, these sub-systems and components may have their set of sub-system and components and create hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

7.6.2 Bottom-Up Approach

The bottom up design model starts with the most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Both, Top-down and Bottom-up approaches are not practical individually. Instead, a good combination of both is used.

Differences between Top Down and Bottom Up Approaches

Some of the differences between top down approach and bottom up approach of software design are described in the table below.

Table 7.1: Differences between Top Down and Bottom Up Approach

Top – Down Approach	Bottom – Up Approach
a) It is breaking down of a program to gain insight into its compositional small programs or modules.	a) It is combining together of modules to give rise to more complex program
b) It begins with high level design and ends with low level design.	b) It begins with low level designs and ends with a high level design
c) The main function is written at first and all the sub-functions are called from the main.	c) Code is developed from modules and then these modules are integrated with the main function.
d) Procedural programming follows top down approach e.g., C, Pascal etc	d) Object oriented programming follows bottom up approach e.g., C++, Java etc.

Software design takes in the user requirements as challenges and tries to find the best solution. While the software is being conceptualized, a plan is being made to find the best possible design for implementing the intended solution.

7.7 VARIANTS OF SOFTWARE DESIGN

The variants of software design are discussed in the following sub sections.

7.7.1 Function Oriented Design

In function-oriented design, the system consists of many smaller sub-systems known as Functions. These functions are capable of performing significant task in the system. The system is considered the top view of all functions.

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 and using the information available globally.

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.
- DFD depicts how functions change the data and the state of the entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

7.7.2 Object Oriented Design (OOD)

Object oriented design (OOD) works around the entities and their attributes instead of functions involved in the software system. This design strategies focus on entities and its attributes. The whole concept of software solution revolves around the engaged entities. Let us now see the important concepts of object oriented design:

- **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 and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes which an object can have and methods, which define the functionality of the object. In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.
- **Encapsulation -** In OOD, the attributes (data variables) and functions (operation on the data) are grouped together is called encapsulation. Encapsulation not only groups data and functions of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance –** Inheritance is a technique of creating a new class from an existing one. It is like copy the existing codes from one location to another location during runtime. In OOD a class is a set of data member and member functions. These members can be inherited from one existing class to another one new class. The existing class is called the base class or parent class and the new class is called the derived class. The derived class may contain some members from base class and some from its own. This reduces the programmer overhead for writing code.
- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but varing in arguments can be assigned the same name. This is called polymorphism. It allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

Design Process

Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented), yet it may have the following steps involved:

- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.

Application framework is defined.

CHECK YOUR PROGRESS

Q6. In object oriented design of software which of the following is false:

a) Objects inherit properties of a class.
b) Classes are defined based on the attributes of the objects.
c) An object can belong to two classes.
d) Classes are always different.

Q7. Choose the option that does not define function oriented software design.

a) It consists of module definitions.
b) Modules represent data abstractions.
c) Modules support functional abstractions.
d) Functions are derived from class.

Q8. The feature of object oriented paradigm that helps in code reuse is _____.

a) Object b) Class
c) Inheritance d) Abstraction

7.8 LET US SUM UP

- Software design is a series of action which is used for convering user requirement into some structure form.
- Software must be made of modules which are independent of each other so that they can be modified and tested separately.
- Cohesion is a measure of the relative functional strength of a module.
- There are seven types of cohesion which are, Coincidental, logical, temporal, procedural, communication, sequential and functional cohesion.
- Coupling is a measure of interconnection among modules.
- There are five types of coupling which are content, common, control, stamp and data coupling.
- Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics.
- Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.
- In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system.
- Object oriented design works around the entities and their characteristics instead of functions involved in the software system.

7.9 FURTHER READINGS

- Mall, R. (2014). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd.
- Sommerville, I. (2013). *Software Engineering: Pearson New International Edition.* Pearson Education Limited.

7.10 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: b) Coupling

Answer to Q2: d) All of the mentioned

Answer to Q3: Modules should not have access to resources which are not required.

Answer to Q4: High cohesion, low coupling

Answer to Q5: a - ii
b - iii
c - i
d - iv

Answer to Q6: c) An object can belong to two classes.

Answer to Q7: d) Functions are derived from class.

Answer to Q8: c) Inheritance

7.11 MODEL QUESTIONS

Q1. What are the attributes of good software?

Q2. Write a short note on modular design.

Q3. Compare Function oriented and Object oriented designs.

Q4. What do you mean by the terms coupling and cohesion in the context of software design? How are these concepts useful in arriving at a good design of a system?

Q5. Differentiate between common and context coupling.

Q6. What is modularity? Explain with an example.

Q7. Give the differences between top down and bottom up approaches of a software design.

Q8. Describe function oriented design.

Q9. Describe object oriented design.

******

UNIT 8: FUNCTION ORIENTED SOFTWARE DESIGN

UNIT STRUCTURE

8.1 Learning Objectives

8.2 Introduction

8.3 Function Oriented Software Design

8.3.1 Entity – Relationship Diagrams

8.4 Decision Trees

8.5 Decision Table

8.6 Data Flow Diagrams (DFD)

8.6.1 Types of DFD

8.6.2 DFD Components

8.6.3 Levels of DFD

8.7 Structure Charts

8.8 Data Dictionary

8.8.1 Requirements of Data Dictionary

8.8.2 Contents of Data Dictionary

8.8.3 Advantages of Data Dictionary

8.9 Pseudocode

8.10 Let Us Sum Up

8.11 Further Readings

8.12 Answers to Check Your Progress

8.13 Model Questions

8.1 LEARNING OBJECTIVES

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

- explain how a software design may be represented as a set of functions
- develop examples which may illustrate the process of function oriented design
- design functional units that transform inputs to outputs
- describe function – oriented design models
- compare sequential and concurrent function – oriented design

8.2 INTRODUCTION

In the previous units, we have discussed software design and the qualities necessary for a good software design along with the concept of modules, cohesion and coupling. Top down and bottom up approach to software design were also covered in the unit in addition

to. The two variant of software design- Function oriented design and object oriented design was introduced in the previous unit.

In this unit we will discuss Function oriented software design in detail. Its definition and the different methods used to represent function oriented software design are covered in this unit. Methods like decision trees, decision tables, data flow diagrams, structure charts, data dictionary and pseudocode. In the next unit, we will explore the concepts related to object oriented design.

8.3 FUNCTION ORIENTED SOFTWARE DESIGN

The function oriented design strategy relies on decomposing the system into a set of functions. It hides the details of an algorithm in the function but the system state information is not hidden. This can cause problem because a particular function can change the state in a way which other functions do not expect. Hence, it will be successful when the amount of system state information is minimized. Systems whose output depends on single input which are not affected by input histories are functionally oriented.

8.3.1 Entity – Relationship Diagrams

Entity-Relationship model is a graphical representation to describe the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them.

ER Model is best used for the conceptual design of database. ER Model can be represented as follows:

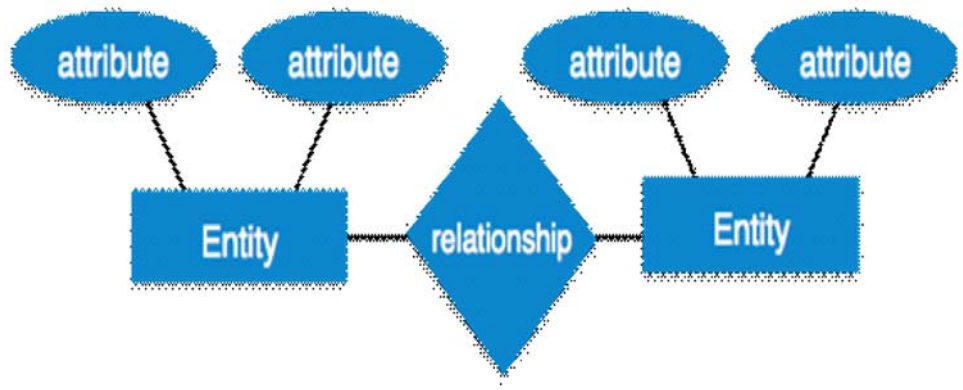


Figure 8.1: Entity – Relationship Model

- **Entity** - An entity in ER Model is a real world being, which has some properties called *attributes*. Every attribute is defined by its corresponding set of values, called *domain*. For example, consider a school database, where student is an entity. Student has various attributes like name, id, age and class etc.
- **Relationship** - The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

Mapping cardinalities:

- one to one
- one to many
- many to one
- many to many

8.4 DECISION TREES

A decision tree is a flowchart like free structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions. The topmost node in a tree is the root node. A typical decision tree is shown in figure 8.2. It is basically used in classification problem. In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node that holds the class prediction for that sample.

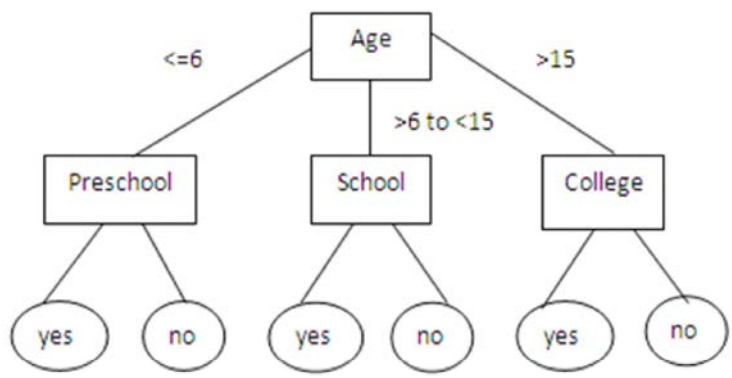


Figure 8.2: Example of Decision Tree

Advantages of decision trees:

- It represents the logic of If-Else in a pictorial form.
- It is useful to express the logic when the value is variable or an action is dependent on nested decision.

- It helps the analyst to identify the actual decision to be made.
- It is used to verify the problems that involve limited number of actions.

Disadvantages of decision trees:

- If there is large number of branches with many paths then it might confuse the analyst rather than helping.

8.5 DECISION TREES

A decision table represents conditions and the respective 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.

Creating Decision Table

To create the decision table, the developer must follow four basic 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.

Example 8.1: Let us take a simple example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while starting the internet and their respective possible solutions. We list all possible problems under column conditions and the prospective actions under column Actions.

Table 8.1: Decision Table example for 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
	Check network cable	X							
	Restart web Browser							X	

Actions	Contact service provider		X	X	X	X	X	X	
	Do no action								
	Check internet router	X				X	X	X	

CHECK YOUR PROGRESS

Q1. An ER model shows the _____.

a. Static view b. Functional view

c. Dynamic view d. All of the above

Q2. What is entity relationship diagram?

Q3. How can we create decision tables?

Q4. What are the advantages and disadvantages of decision trees?

8.6 DATA FLOW DIAGRAMS (DFD)

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

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

8.6.1 DFD Components

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



Figure 8.3: DFD Components

- **Entities** - Entities are source and destination of information data. Entities are represented by rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage** - 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** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

8.6.2 Levels of DFD

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

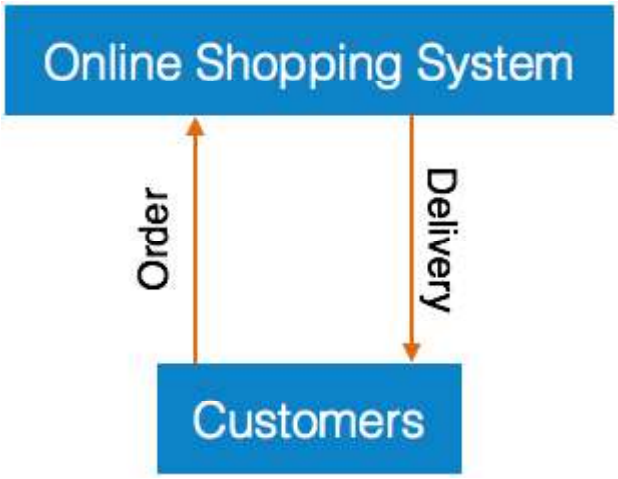


Figure 8.4: Example of Level 1 DFD

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

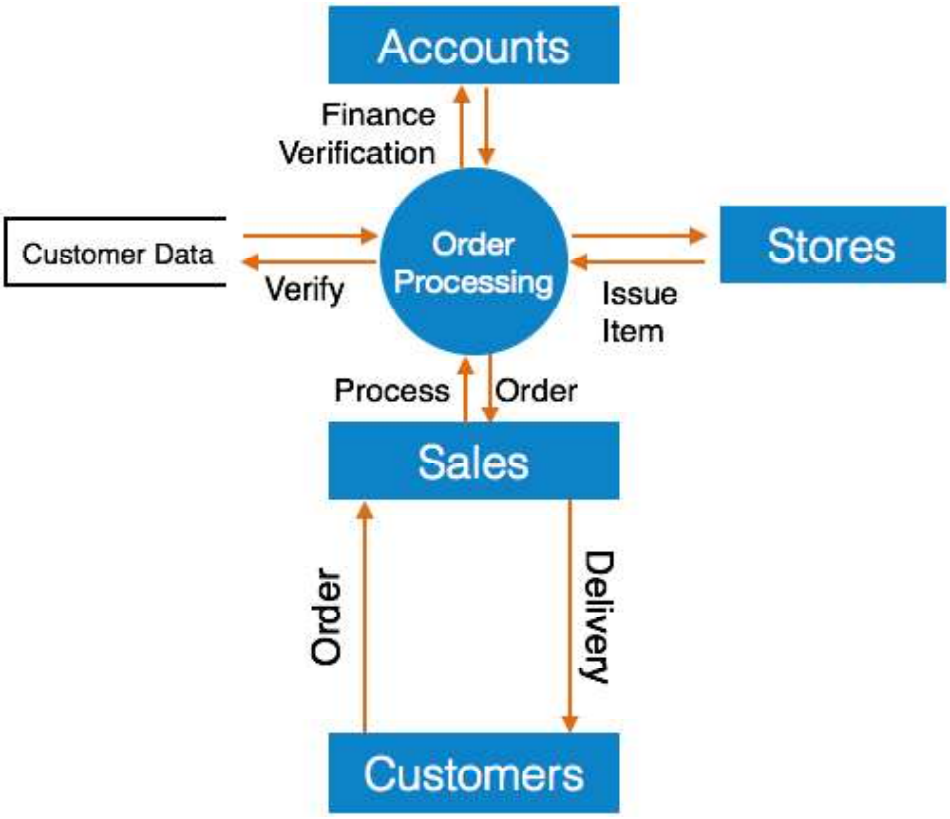


Figure 8.5: Example of Level 2 DFD

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

8.7 STRUCTURE CHARTS

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD. Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

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 can be invoked from many modules.

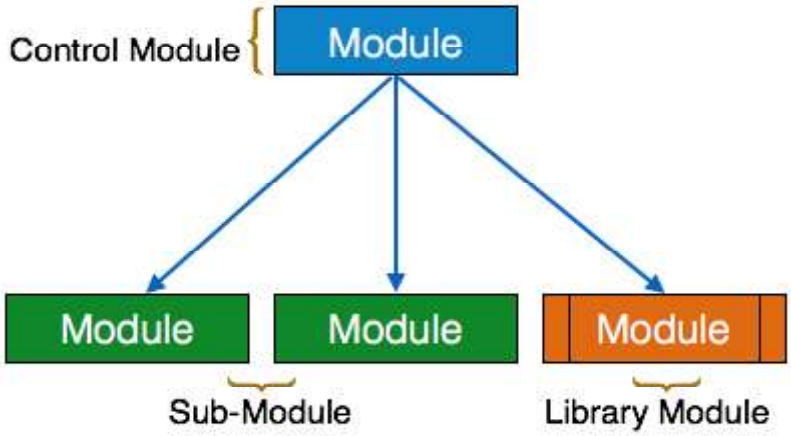


Figure 8.6: Example of Module

- **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.

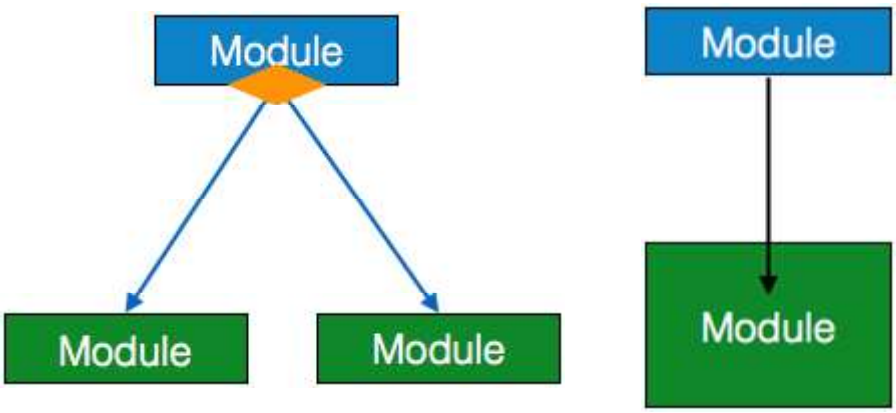


Figure 8.7: Example of (a) Condition in Module (b) Jump in Module

- **Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.

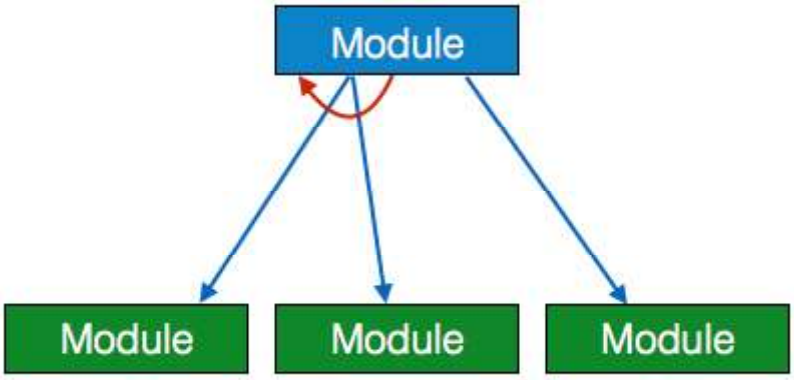


Figure 8.8: Example of Loop in Module

- **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.

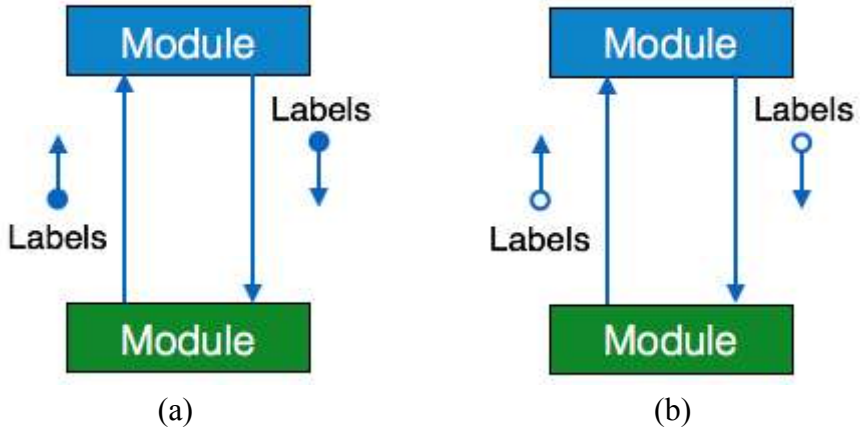


Figure 8.9: Example of (a) Data Flow in Module (b) Control Flow in Module

Analysts and designers of the software have come up with tools such as Structured English which consists of the description of what is required to code and how to code it. Structured English helps the programmer to write error-free code. Structured English uses plain English words in structured programming paradigm. It is not the ultimate code but a kind of description of what is required to code and how to code it. The following are some examples of structured programming.

IF-THEN-ELSE

DO-WHILE-UNTIL

Analyst uses the same variable and data name, which are stored in Data Dictionary, making it much simpler to write and understand the code.

Example 8.2: We take the same example of Customer Authentication in the online

shopping environment. This procedure to authenticate customer can be written in Structured English as:

```
Enter Customer_Name
Seek Customer_ Name in Customer_DB file
IF Customer_Name found THEN
        Call procedure USER_PASSWORD_AUTHENTICATE()
Else
        PRINT error message
        Call procedure NEW_CUSTOMER_REQUEST()
ENDIF
```

The code written in Structured English is more like day-to-day spoken English. It cannot be implemented directly as a code of software. Structured English is independent of programming language.

8.8 DATA DICTIONARY

Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc. Data dictionary has rigorous definitions of all names in order to facilitate the user and software designers.

Data dictionary is often referenced as **meta-data** (data about data) repository. It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

8.8.1 Requirement of Data Dictionary

The data is referenced via data dictionary while designing and implementing software. Data dictionary removes any chances of ambiguity. It helps keeping work of programmers and designers synchronized while using the same object reference everywhere in the program. Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

8.8.2 Contents of Data Dictionary

Data dictionary should contain information about the following:

a) Data Flow
b) Data Elements
c) Data Stores

d) Data Processing

a) Data Flow

Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described below:

Table 8.2: Data Flow elements

=	Composed of
{}	Repetition
()	Optional
+	And
[/]	Or

Example: Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

b) Data Elements

Data elements consist of name and descriptions of data and control Items, internal or external data stores etc. with the following details: Primary Name, Secondary Name (Alias), Use-case (How and where to use), Content Description (Notation etc.), supplementary information (preset values, constraints etc.)

c) Data Store

It stores the information from where the data enters into the system and exists out of the system. The data store may include files and tables. Files are Internal to software. It can be external to software but it should be on the same machine or external to system located on different machine. Tables are used for naming convention and indexing property.

d) Data Processing

There are two types of data processing. They are mentioned below:

Logical: As user sees it

Physical: As software sees it

Advantages of Data Dictionary

- It gives well-structured and clear information about the database.
- One can analyze if there is any redundancy like duplicate columns, tables etc.
- It is helpful for a new database administrator to understand database.
- Since database is very huge and will have lots of tables, views, constraints, indexes etc, it will be difficult for anyone to remember. Data dictionary helps user by providing these details.

8.9 PSEUDOCODE

The narrative description of a computer program in English language is called Pseudocode. Pseudo code avoids variable declaration but they are written using structured English. Pseudo code provides a method to perform the task, as if a computer is executing the code.

Example 8.3: Pseudocode for addition of two numbers

Step 1: Read A, B

Step 2: Compute C = A + B

Step 3: Print C

Step 4: Stop

CHECK YOUR PROGRESS

Q5. Which tool is used for structured designing?

a) Program flowchart b) Structure chart

c) Data Flow Diagram d) Module

Q6. At the highest level, a DFD is referred to as _____.

a) Scope Diagram b) Context Diagram

c) Level 1 DFD d) Level 2 DFD

Q7. In a DFD, an originator or data receiver is usually designated by

a) a square box b) a circle

c) a rectangle d) none of these

Q8. A zero level DFD describes

a) a full system design

b) data flow in all the modules

c) overview of processes inputs and outputs

d) none of these

Q9. Pseudocode is described in _____.

a) Programming language b) C Language

c) English Language d) All of the above

Q10. In a DFD, user interaction with the system is denoted by __

a) Circle b) Arrow

d) Rectangle d) Triangle

Q11. Which of the following gives a logical structure of the database graphically?

a) ER Diagram b) DFD

c) Structure Chart d) None of these

Q12. A DFD depicts _____

a) Flow of data b) Flow of control

c) Both a) and b) d) None of these

Q13. Pseudo code can replace ____

a) Flowchart b) Structure Chart

c) Decision Table d) Graphs

Q14. If you were collecting and storing information about your music collection, an album would be considered a _____

a) Relation b) Entity

c) Instance d) Attribute

Q15. What term is used to refer to a specific record in your music database for example, information stored about a specific album?

a) Relation b) Instance

c) Table d) Column

Q16. Structure Charts are the products of _____

a) Requirement gathering b) Requirement analysis

c) Design d) Coding

8.10 LET US SUM UP

- The function oriented design strategy relies on decomposing the system into a set of interacting functions.
- Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them.
- A decision tree is a graphical representation that presents conditions and actions sequentially. It is a method of showing the relationship of each condition and its permissible actions.
- A decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.
- Data flow diagram is graphical representation of flow of data in an information system.

- Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD.
- Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc.
- Pseudo code is written more close to programming language. It may be considered as programming language, full of comments and descriptions.

8.11 FURTHER READINGS

- Elmasri, R. (2008). *Fundamentals of database systems.* Pearson Education India.
- Kahate, A. (2004). *Introduction to Database Management Systems.* Pearson Education India.

8.12 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: a) Static view

Answer to Q2: Entity Relationship Diagram is a graphical representation of the object and the relationship pair. It is mainly used in database application.

Answer to Q3: We can create decision table as follows:

- Identify all possible conditions to be addressed
- Determine actions for all identified conditions
- Create Maximum possible rules
- Define action for each rule

Answer to Q4: Advantages of decision trees:

a) It represents the logic of If-Else in a pictorial form.
b) It is useful to express the logic when the value is variable or an action is dependent on nested decision.
c) It helps the analyst to identify the actual decision to be made.
d) It is used to verify the problems that involve limited number of actions.

The disadvantage of a decision tree is that if there is large number of branches with many paths then it might confuse the analyst rather than helping.

Answer to Q5: b) Structure Chart

Answer to Q6: b) Context Diagram

Answer to Q7: c) A rectangle

Answer to Q8: c) Overview of processes inputs and outputs

Answer to Q9: c) English language

Answer to Q10: b) Arrow

Answer to Q11: a) ER Diagram

Answer to Q12: a) Flow of data

Answer to Q13: a) Flowchart

Answer to Q14: b) Entity

Answer to Q15: b) Instance

Answer to Q16: c) Design

8.13 MODEL QUESTIONS

Q1. Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

Q2. Construct an ER diagram of a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of various tests and examination conducted.

Q3. What is a DFD? Draw a DFD for a library management system.

Q4. Explain the various notations used in Data Flow Models.

Q5. What is the need for a data dictionary?

Q6. What is a structured chart? What are the elements of a structured chart?

Q7. Differentiate between Structured Chart and Flowchart.

Q8. Differentiate between Structured English and Pseudocode.

****

UNIT 9: OBJECT ORIENTED SOFTWARE DESIGN

UNIT STRUCTURE

9.1 Learning Objectives
9.2 Introduction
9.3 Object Oriented Software Design
9.3.1 Object-Oriented Concepts
9.4 Unified Modeling Language (UML)
9.5 Modeling Types
9.6 Basic Notations and Diagrams
9.7 Class Diagrams
9.8 Use Case Diagrams
9.9 Activity Diagrams
9.10 Let Us Sum Up
9.11 Further Readings
9.12 Answers to Check Your Progress
9.13 Model Questions

9.1 LEARNING OBJETIVES

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

- describe Unified Modeling Language (UML)
- describe object oriented concepts
- describe basic notations and diagrams
- draw class diagrams, use case diagrams and activity diagrams

9.2 INTRODUCTION

In the previous units, we have discussed the function oriented software design in detail. The definition and the different methods used to represent function oriented software design were included in the discussion along with the definition and different methods used for representing function oriented software design. Methods like decision trees, decision tables, data flow diagrams, structure charts, data dictionary and pseudocode have been described in detail in the previous unit.

In this unit we will discuss object oriented software design in detail. You will learn the basic notations and diagrams used in UML which will help us in building the foundation towards designing software products based on OOD approach. Class diagrams, use case diagrams and activity diagrams dictionary are also described in detail in this unit. In the next unit, we will explore the concepts related to software testing.

9.3 OBJECT ORIENTED SOFTWARE DESIGN

A good software product can be expected if it is followed by a good design. The basic purpose of software-design is to formalize the desired requirements of the software product and to specify these technically. With the ever changing list of user requirements a software product is always estimated to grow continuously. A good software design should be highly concerned with this continuous evolution of a software product. In earlier days procedural paradigm was followed in order to design software products. But this approach became obsolete to meet the requirements of products with larger in size and increased complexity. Out of the many alternatives, object oriented paradigm has been considered to be the most appropriate and popular one. Object Oriented Design (OOD) aims to design a software product based on some objects determined during the analysis phase. For OOD a common language is required to be understood. Unified Modeling Language (UML) is the most popular formalism for this purpose.

9.3.1 Object-Oriented Concepts

Object-Oriented Analysis and Design is defined as the investigation of objects and their collaboration, to form fully functional software systems. UML is generally referred to as the successor of object-oriented (OO) analysis and design. The purpose of object-oriented analysis and design are –

- To identify the objects of a system
- To identify the relationships between them
- To create a design of the system, which can be converted to a real world executable environment using object-oriented languages
- Polymorphism: Mechanism of the same entity existing in different forms

9.4 UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling Language (UML) is a graphic based language for visualizing, specifying, documenting and constructing the artifacts of a software-intensive system. UML was designed by the Object Management Group abbreviated as OMG and UML 1.0 specifications were drafted by the same in January 1997.

- UML is not the same as other common programming languages like C++, Java, and COBOL etc.
- UML is a pictorial language most commonly used for designing software blueprints.
- Modelling software systems is not the only use of UML, it can also be used to model non-software systems, for instance the process flow in a manufacturing unit etc.

CHECK YOUR PROGRESS

Q1. When UML was first released?

Q2. UML was designed by _____.

Q3. Name the four main properties of Object Oriented Design.

Q4. _____ is generally referred to as the successor of OOD.

9.5 MODELLING TYPES

In UML, different types of diagrams are used to model different phases of the software system. They can be classified into three types.

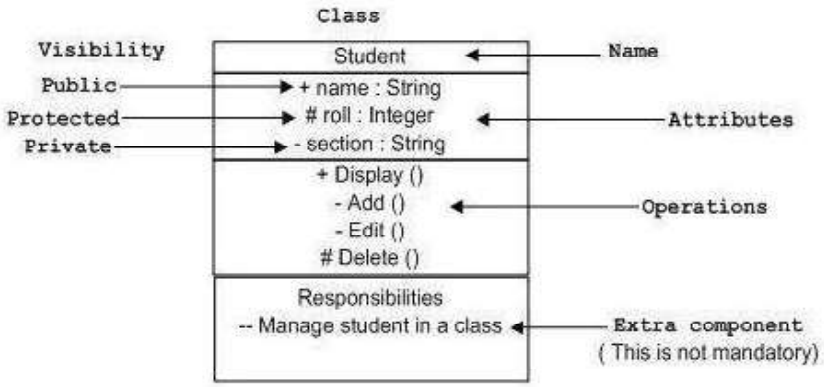
- **Structural Modelling:** This type of modelling summarizes the static features of a software system. They are: Class Diagrams, Objects Diagrams, Deployment Diagrams, and Package Diagrams etc.
- **Behavioral Modelling:** This type of modelling describes the interaction among structural diagrams, showing the dynamic nature of the software system. They are- Activity Diagrams, Use Case diagrams and Interaction Diagrams.
- **Architectural Modelling:** This represents the overall framework of the software-intensive system. It contains both structural and behavioral aspects of the system.

9.6 BASIC NOTATIONS AND DIAGRAMS

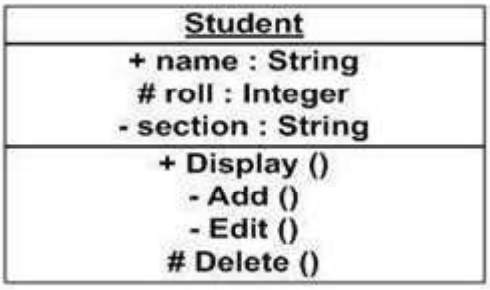
UML is very popular because of its easy to use diagrammatic notations. UML notations are one of the most important elements in software system modelling. Appropriate and efficient use of these notations is of utmost importance for making a meaningfully complete software-system model.

Some of the commonly used notations are shown below.

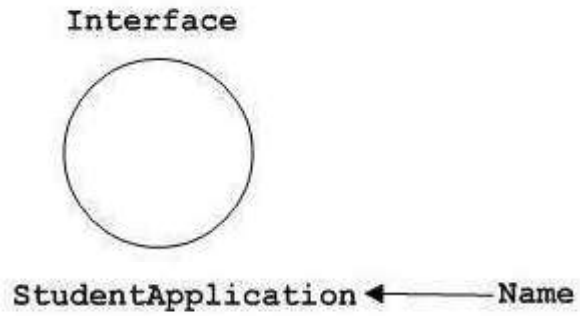
Class Notation:



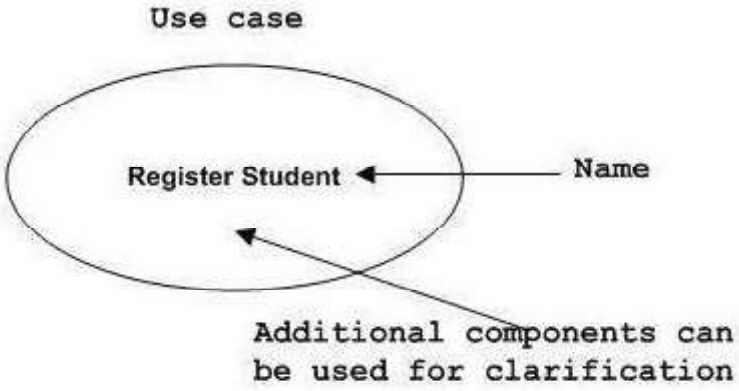
Object Notation:



Interface Notation:



Use Case Notation:



Actor Notation:



Figure 9.1: Basic Notations

All these notational elements come together to form a meaningful representation.

9.7 CLASS DIAGRAMS

Class diagrams are static representations of software systems. It is generally used for constructing executable object oriented code for the software application. It describes the attributes or properties of a class and the operations to perform on the instances of the class. Class diagrams show a coming together of classes, interfaces, collaborations, associations and constraints, also known as a structural diagram altogether.

The procedure for drawing a class diagram:

- Name of the class diagram is generally the meaningful summary of the system or project as a whole.
- Relationships between each element must be defined
- Attributes and methods for each class are identified and formalized
- All necessary properties of the classes involved should be properly defined

An example of a class diagram of an Order System is shown below.

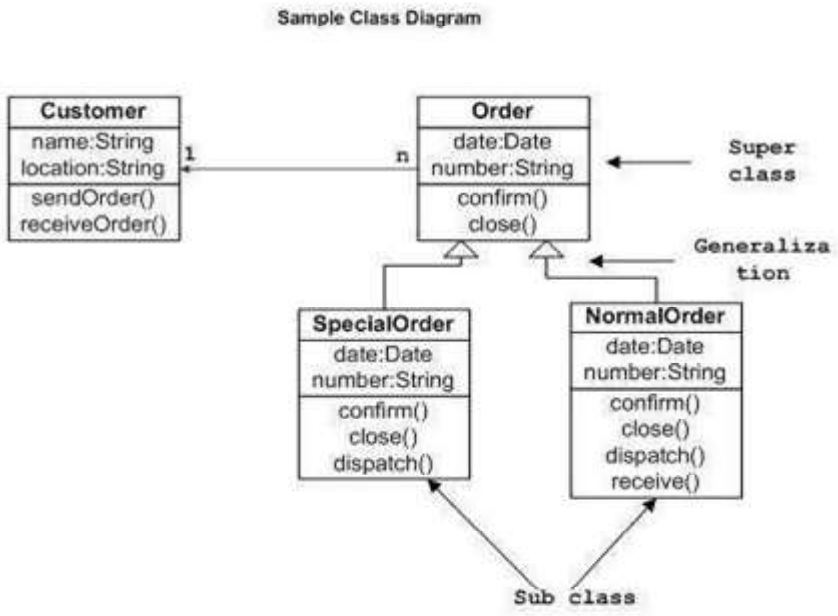


Figure 9.2: Example of Class diagram for Order System

CHECK YOUR PROGRESS

Q5. Name the three different types of modeling in UML.

Q6. _____ modeling represents the overall framework of the software-intensive system.

Q7. Draw a Class Diagram of a Loaning system in a bank which checks the applier's savings before granting loan.

Q8. Class diagrams provide a _____ view of the system.

9.8 USE CASE DIAGRAMS

The graphical representation of the dynamic aspect and properties of a software system is done through Use Case diagrams. Dynamic behavior of a system means its properties while it is running or operating. Dynamic behavior and its modeling of a system is considered, in many cases, to be more important than static behavior.

The internal and external factors that interact with the system and bring changes to its data or properties are knows as Actors. Use Case diagrams are representations of these Actors, Use Cases and the Relationship between them when the system is operational.

The general procedure for drawing a Use Case diagram is as follows:

- The name of the diagram must identify the main functionality of the system
- Identify the Actors involved and give them suitable names
- Show the relationship between the elements of the diagram

As an example, the Use Case diagram representing an order management system is shown below.

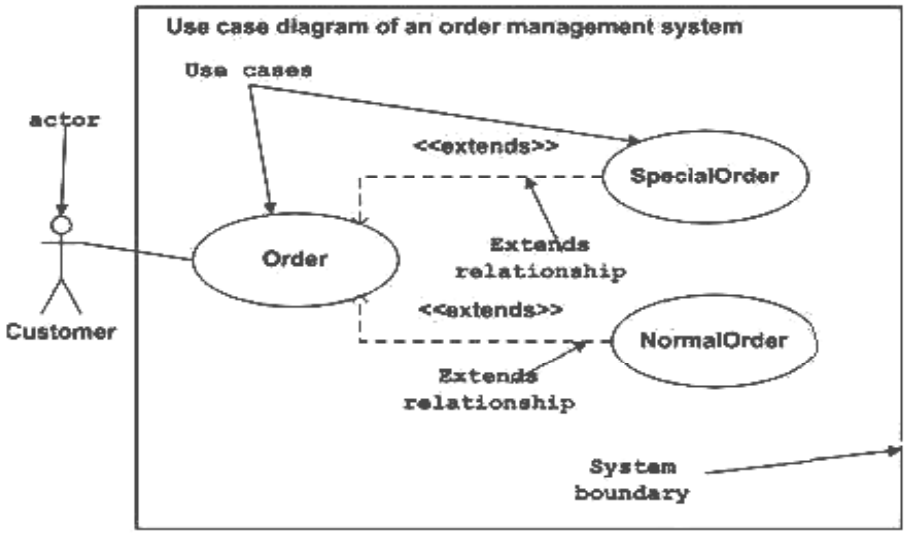


Figure 9.3: Example of Use Case Diagram

9.9 ACTIVITY DIAGRAMS

An activity diagram is yet another graphical method of representing the dynamic aspects of a software system. It is basically a flowchart to showcase the flow on events, from one activity to another. An event can be described as an operation that occurs in the software system.

The steps to create an activity diagram are as follows:

- Identify the following in the system:
  - Activities or events
  - Associations between events
  - Conditions for events to occurs
  - Constraints
- Create links between each event or activity so that each possible flow of actions or activities in the software system is present in the diagram

The following is an example of an activity diagram of an Order Management system. It shows the occurrence of four main activities: Sending orders by customer, receipt of orders, Confirmation of orders and dispatch of orders.

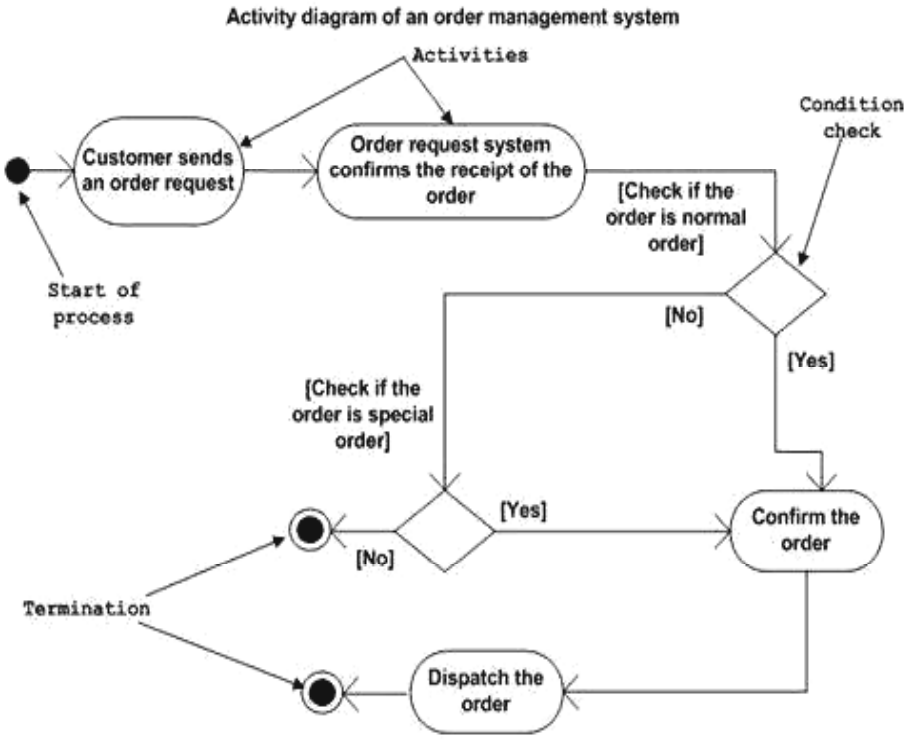


Figure 9.4: Example of Activity Diagram

CHECK YOUR PROGRESS

Q9. Draw the Use Case diagram of a simple payment system with a user and one administrator and a clerk.

Q10. All the internal and external factors in a Use Case diagram are called _____.

9.10 LET US SUM UP

- The Unified Modelling Language (UML) is designed to be a language of graphic means for visualizing, specifying, documenting and constructing the artifacts of a software-intensive system.
- Object-Oriented Analysis and Design is defined as the investigation of objects and their collaboration, to form fully functional software systems.
- The types of UML modeling are- Structural, Behavioral and Architectural.
- UML diagrams consist of many basic notational elements like: Class, Object, Interface, Actor etc.
- Class diagrams are static representations of software systems. It is generally used for constructing executable object oriented code for the software.
- The graphical representation of the dynamic aspect and properties of a software system are done through Use Case diagrams.
- Activity diagram is basically a flowchart to showcase the flow on events, from one activity to another, in a software-intensive system.

9.11 FURTHER READINGS

- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. E. (1991*). Object-oriented modeling and design* (Vol. 199, No. 1). Englewood Cliffs, NJ: Prentice-hall.
- Chonoles, M. J., & Quatrani, T. (1996). *Succeeding with the Booch and OMT methods: a practical approach*. Addison-Wesley.

9.12 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: January 1997

Answer to Q2: Object Management Group

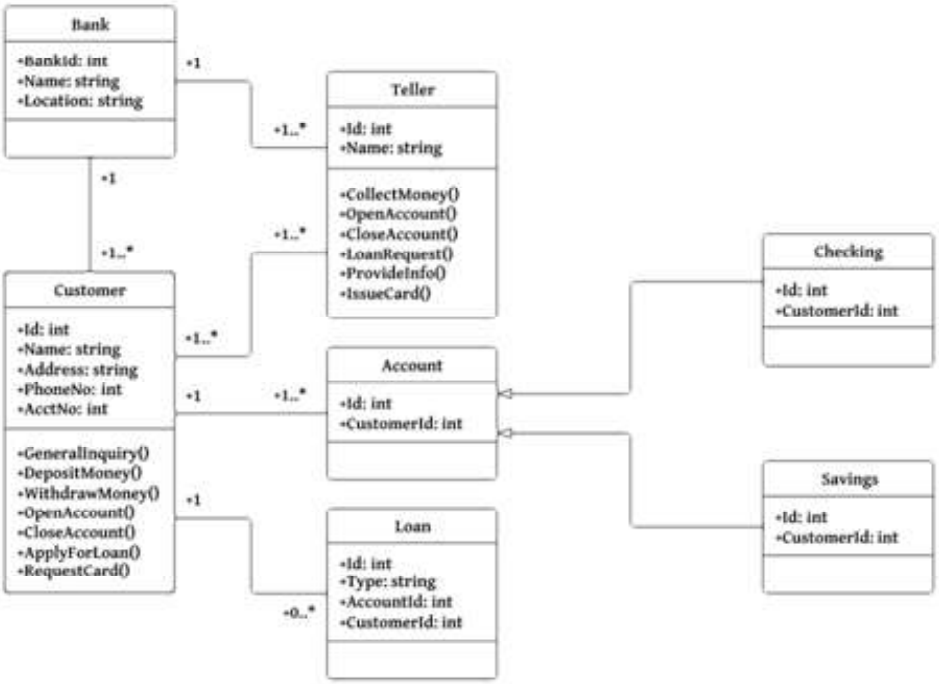
Answer to Q3: Abstraction, Encapsulation, Inheritance and Polymorphism

Answer to Q4: UML

Answer to Q5: Structural, behavioral and architectural

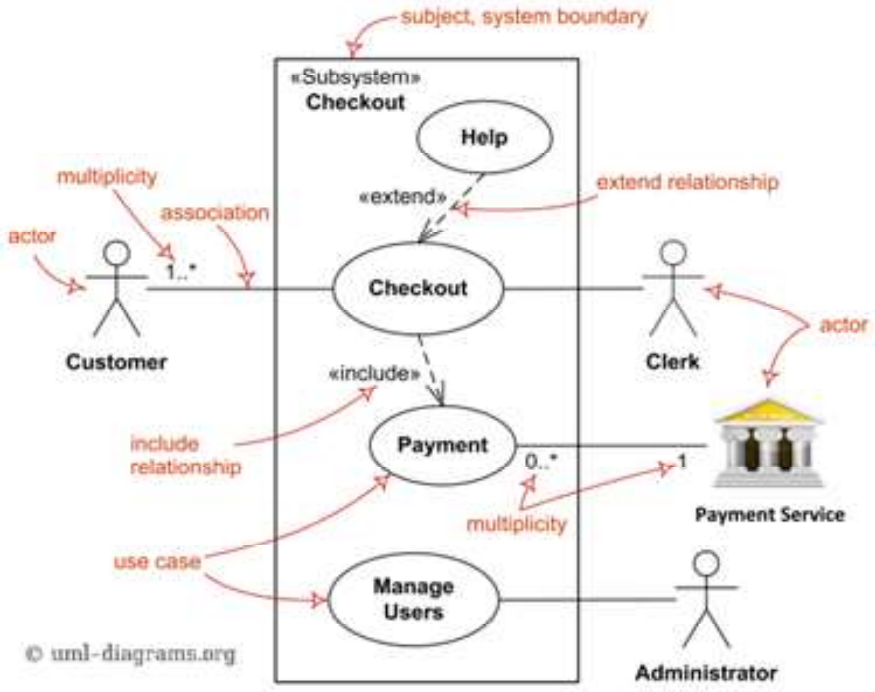
Answer to Q6: Architectural

Answer to Q7:



Answer to Q8: Static

Answer to Q9:



9.13 MODEL QUESTIONS

Q1. Define UML.

Q2. How is UML relevant to OO Analysis and Design?

Q3. Define OO Analysis and Design.

Q4. What is the purpose of OO Analysis and Design?

Q5. Define the four basic concepts of OO approach.

Q6. Draw the class diagram of an ATM, with both withdraws and deposit facilities.

Q7. Draw the Use Case diagram of the same in question 6, with actors being user and bank.

Q8. Draw an Activity diagram of the system described in question 6.

Q9. Differentiate between Use Case diagrams and Activity diagrams.

Q10. Differentiate between Use Case diagrams and Class diagrams.

******

UNIT 10: USER INTERFACE DESIGN

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 User Interface Design

10.3.1 Characteristics of a Good User Interface

10.4 Concept of User Interface Design

10.4.1 User Guidance and Online Help

10.4.2 Modeless Interface

10.4.3 Graphical User Interface and Test-based User Interface

10.4.4 Types of User Interfaces

10.5 Fundamentals of Component-based GUI development

10.6 User Interface Design Methodology

10.7 Let Us Sum Ups

10.8 Answers to Check Your Progress

10.9 Further Reading

10.10 Model Questions

10.1 LEARNING OBJECTIVES

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

- understand the characteristics of a good user interface.
- learn the basic concepts of user interface design, including user guidance and online help, modeless interfaces, graphical user interfaces versus text-based interfaces, and different types of user interfaces.
- acquire knowledge of the fundamentals of component-based GUI development.
- understand the methodology of user interface design.

10.2 INTRODUCTION

In the previous unit, we learned about object-oriented software design in detail. We learned about unified modelling language(UML) and UML diagrams along with use case models, class models and activity diagrams. In this unit, we will learn about the characteristics of a good user interface design. The basic concept regarding user guidance and online help is also described in this unit. The unit also describes in detail the fundamentals of component-based GUI development and user interface design methodology. In the next

unit, we will explore the concepts related to software coding like code review, code inspection and software documentation.

10.3 USER INTERFACE DESIGN

A User Interface (UI) is the point at which a user interacts with a computer system, software application, or digital device. It consists of all the elements that enable communication between the user and the system, such as menus, buttons, icons, windows, and commands. In simple terms, the user interface services as a bridge between the user and the computer system, allowing users to give input and receive output. A well-designed interface helps users complete tasks easily, quickly, and effectively.

User Interface Design is the process of creating the visual layout and interactive elements of a system to ensure that users can interact with it in an effective, efficient, and satisfying way. It focuses on:

- Improving usability
- Enhancing the user experience
- Making systems easy to learn and use

The primary objectives of UI design are:

- Ease of Use: The system should be simple and user-friendly.
- Efficiency: Users should be able to complete tasks quickly.
- Consistency: Similar actions should behave in the same way across the system.
- Error Reduction: The design should minimize the chances of user errors.
- User Satisfaction: The interface should offer a pleasant experience.

User interface design is important because it:

- Enhance interaction between users and systems.
- Increase productivity and efficiency.
- Reduces the training time for new users.
- Minimizes the chances of user errors.
- Improves overall user satisfaction.

For example, a well-designed ATM interface, mobile app, or website enables users to perform tasks such as withdrawing money, sending messages, or shopping online quickly and easily.

10.3.1 CHARACTERISTICS OF A GOOD USER INTERFACE DESIGN

The key characteristics of a good user interface are described below:

- **Simplicity*:*** A good user interface should be clear and easy to understand. Users should be able to perform tasks without confusion. The design should avoid unnecessary elements and focus only on essential features.

- **Consistency:** The interface should be uniform in design, layout, colors, fonts, and commands across the system. Similar actions should work the same way in all parts of the application, helping users learn and remember its usage easily.
- **User Friendliness:** A good UI should be comfortable and easy to use. It should cater to users of all skill levels, from beginners to experienced users.
- **Clarity*:*** All buttons, icons, menus, and instructions should be clearly labelled and easy to understand. Information on the screen should be well-organized and meaningful.
- **Feedback*:*** The system should always provide feedback on user actions. For example, it should display a confirmation message when a file is saved or a task is completed.
- **Efficiency*:*** A well-designed interface enables users to complete tasks quickly with minimal effort. Frequently used options should be easily accessible.
- **Error Prevention and Handling:** The interface should help users avoid mistakes by providing proper guidance. It errors occur, messages with clear meaning should be shown along with suggestions to fix them.
- **Flexibility:** A good UI should offer multiple ways to perform tasks (such as menus, shortcuts, or commands), allowing users to choose their preferred method.
- **Accessibility*:*** The interface should be usable by everyone, including people with disabilities. Features like screen readers, larger text, and high-contrast modes can support this.
- **Aesthetic Design*:*** An attractive interface with a good layout, appropriate colors, and clear typography enhances the user experience and makes the system more engaging.

CHECK YOUR PROGRESS

Q1. Choose the correct answer:

a) Which of the following is a key characteristic of a good user interface?
   i) Complexity ii) Consistency iii) Ambiguity iv) Redundancy

b) A good user interface should primarily focus on:
   i) Developer convenience ii) System performance only
   iii) User needs and experience iv) Hardware limitations

c) Which characteristic ensures that similar actions produce similar results?
   i) Flexibility ii) Consistency iii) Efficiency iv) Portability

d) Providing feedback to users after an action is important because it:
   i) Slows down the system ii) Confuses the user

iii) Keeps users informed about system status iv) Reduces usability

e) Which of the following refers to how easily a new user understands and uses the interface?

i) Reliability ii) Learnability iii) Scalability iv) Maintainability

Q2. Fill in the blanks:

a) A good user interface should be simple, clear, and easy to _____ for users.

b) The _____ of a user interface ensures that similar actions work the same way throughout the system.

c) The system should provide _____ to inform users about the result of their actions.

d) A well-designed interface improves _____ by allowing users to complete tasks quickly with minimal effort.

e) The interface should include features like screen readers and high-contrast modes to improve _____.

Q3. Mention any two characteristics of a good user interface and explain briefly.

10.4 CONCEPT OF USER INTERFACE DESIGN

The basic concept of User Interface (UI) design refers to the fundamental ideas and principles used to create interfaces that allow users to interact effectively with a computer system or application. The main goal of UI design is to make the interaction between the user and the system simple, efficient, and user-friendly.

Below are the important basic concepts of user interface design:

1. **User-Centred Design:** UI design should focus on the needs, preferences, and abilities of the users. The system should be designed from the user's point of view rather than the developers.

2. **Usability:** Usability means how easy and efficient it is for users to learn and use the system. A usable interface allows users to complete tasks with minimal effort and confusion.

3. **Consistency:** Consistency in layout, colors, fonts, icons, and commands helps users understand and remember how to use the system. Similar actions should behave in the same way throughout the interface.

4. **Feedback:** The system should always inform users about what is happening. For example, when a task is completed or an error occurs, the interface should display a message or visual response.

5. **Simplicity**: A good interface should be simple and free from unnecessary elements. A clean and uncluttered design helps users focus on important tasks.

6. **Accessibility:** The interface should be designed so that all users, including people with disabilities, can use it. This includes support for screen readers, large text, and high-contrast displays.
7. **Efficiency*:*** A well-designed UI should enable users to complete tasks efficiently, with frequently used features being easily discoverable and readily accessible.
8. ***Error Prevention and Recovery:*** A good interface should prevent errors by guiding users properly. If mistakes happen, the system should provide clear error messages and allow users to correct them easily.
9. ***Visual Hierarchy:*** Important information and controls should stand out more than less important ones. This helps users quickly understand where to look and what to do first.
10. ***Aesthetic and Visual Design:*** The interface should be visually appealing with proper use of colors, fonts, spacing, and layout. A pleasant design improves user satisfaction and engagement.

These basic concepts together help designers create interfaces that are easy to learn, efficient to use, and enjoyable for users.

10.5.1 USER GUIDANCE AND ONLINE HELP

User guidance and online help are essential components of a software system that support users in understanding and using the application effectively. They provide instructions, explanations, and assistance during user interaction with the system.

USER GUIDANCE: It refers to the support provided to help users understand how to operate a system or application. Its main goal is to reduce confusion and improve the efficiency of user interaction.

Features of User Guidance:

1) ***Instructions and Prompts*:** The system provides clear directions or prompts to guide users on what to do next.
2) ***Tooltips and Hints*:** When a user moves the cursor over a button or icon, a small message (tooltip) appears explaining its function.
3) ***Error Messages*:** If a mistake occurs, the system displays clear and meaningful error messages along with possible solutions.
4) ***Input Validation*:** The system checks the correctness of user input and prevents incorrect data entry.
5) ***Step-by-Step Assistance*:** Some systems provide guided steps or wizards to help users complete tasks easily.

Importance of User Guidance

- Helps new users learn the system quickly
- Reduces errors during system use

- Improves efficiency and productivity
- Enhances user satisfaction

ONLINE HELP: It is a built-in help system available within the software that users can access whenever they need assistance. It provides detailed explanations about features and operations.

Types of Online Help

1) ***Context-Sensitive Help*:** Provides help related to the specific task or screen the user is currently working on.
2) ***Help Menu / Help Files*:** A structured set of help documents that explain the features and functions of the system.
3) ***Tutorials*:** Step-by-step guides that teach users how to perform tasks in the software.
4) ***Frequently Asked Questions (FAQs)*:** A list of common user questions and their answers.
5) ***Searchable Help System*:** Allows users to search for specific topics or problems within the help documentation.

Advantages of Online Help

- Available instantly within the software
- Easy to access and use
- Provides detailed explanations and examples
- Helps users solve problems without external support

10.5.2 MODELESS INTERFACE

A modeless interface is a type of user interface that allows the user to interact with different parts of the application without being restricted to a particular mode. In this interface, the user can perform multiple actions or switch between tasks freely.

Definition: A modeless interface is an interface in which the user can access and use different functions of a program at any time without completing or closing the current operation.

Example: A common example is Microsoft Word. While working on a document, users can open the Find and Replace dialogue box, check spelling, or access other menu options without interrupting or closing the current task.

Features of Modeless Interface

1) ***Freedom of Interaction*:** Users can move between different windows, tools, or commands without restrictions.

2) ***No Fixed Mode*:** The system does not force the user to remain in a particular mode to perform an operation.
3) ***Multiple Tasks*:** Users can perform several tasks simultaneously within the same application.
4) ***Continuous Access*:** Other options and commands remain available while working on a task.

Advantages of Modeless Interface

- Flexible to use.
- Allows multitasking.
- Improves productivity.
- User can switch tasks easily.

Disadvantages of Modeless Interface

- May sometimes confuse inexperienced users.
- Users may accidentally perform unintended actions.

10.5.3 GRAPHICAL USER INTERFACE VS TEXT-BASED USER INTERFACE

A user interface (UI) is the medium through which a user interacts with a computer system. Two common types of interfaces are the Graphical User Interface (GUI) and the Text-Based User Interface (TUI). Let us look at the differences in aspects for both GUI and TUI in the table below.

Aspect	**Graphical User Interface (GUI)**	**Text-Based User Interface (TUI)**
Definition	A GUI allows users to interact with a computer using graphical elements such as windows, icons, menus, and buttons.	A TUI allows users to interact with a system by typing commands using the keyboard.
Interaction Method	Uses mouse, touch, or keyboard.	Mainly uses keyboard commands.
Ease of Use	Easy to learn and us er-friendly, especially for beginners.	More difficult for beginners because commands must be memorized.
Visual Elements	Uses graphics, icons, menus, and dialog boxes.	Uses text and command lines only.
Speed	Sometimes slower due to graphical processing.	Often faster for experienced users who know the commands.

Speed	Sometimes slower due to graphical processing.	Often faster for experienced users who know the commands.
System Resources	Requires more memory and processing power.	Requires fewer system resources.
Error Handling	Provides visual feedback and error messages.	Errors occur if commands are typed incorrectly.
Examples	Operating systems such as Microsoft Windows and macOS use GUI.	Command-line environments such as MS-DOS and Unix Shell use text-based interfaces.

Table 10.1: Differences in aspects for both GUI and TUI

Advantages of GUI

- Easy to use and understand.
- No need to remember commands.
- Visual representation of tasks.
- Suitable for beginners.

Advantages of Text-Based Interface

- Faster for experienced users.
- Uses fewer system resources.
- More control through commands.

10.4.4 TYPES OF USER INTERFACE

A user interface **(UI)** is the medium through which users interact with a computer system or software application. Different types of user interfaces are designed to make interaction easier depending on the needs of users and the nature of the system.

1) **Graphical User Interface (GUI):** A Graphical User Interface (GUI) allows users to interact with a computer using graphical elements such as windows, icons, menus, and buttons. Users typically use a mouse, keyboard, or touch to operate it.
   Examples: Operating systems such as Microsoft Windows and macOS use graphical interfaces.
   Features:
   - Uses visual elements like icons and graphics.
   - Easy to learn and user-friendly.
   - Supports point-and-click interaction.
2) **Text-Based User Interface (TUI) / Command Language-Based Interface (CLI):** A text-based user interface allows users to interact with the system by typing commands using the keyboard.

Examples: Systems like MS-DOS and UNIX Shell use command-line interfaces.

Features:

- No graphical elements
- Requires knowledge of commands
- Uses fewer system resources

3) **Menu-Driven Interface:** In a menu-driven interface, the user selects options from a list of menus displayed on the screen.
   Examples: Interfaces used in ATM machines and many application menus.
   Features:
   - Easy for beginners
   - Does not require remembering commands
   - Options are displayed in structured menus
4) **Form-Based Interface:** A form-based interface allows users to enter data into fields arranged in a form.
   Example: Online registration forms and database entry systems.
   Features:
   - Used for data entry and record management
   - Structured format for input
   - Easy validation of data
5) **Natural Language Interface:** A Natural Language Interface (NLI) allows users to interact with the system using human language (spoken or written).
   Example: Voice assistants such as Siri and Google Assistant.
   Features:
   - Very user-friendly
   - Uses voice or natural text input
   - Requires advanced language processing
6) **Touch-Based Interface:** A touch interface allows users to interact with devices by touching the screen directly.
   Example: Smartphone interfaces such as Android.
   Features:
   - Common in smartphones and tablets
   - Supports gestures like tap, swipe, and pinch
   - Very intuitive for users

CHECK YOUR PROGRESS

Q4. Fill-in-the-blanks:

a) UI design should focus on the needs and preferences of the users; this approach is called _____ design.

b) _____ refers to how easy and efficient it is for users to learn and use a system.

c) A clean and uncluttered interface reflects the principle of _____ in UI design.

d) _____ refers to the support provided to help users understand how to operate a system or application.

e) Small messages that appear when a user moves the cursor over an icon are called _____.

f) _____ help provides assistance related to the specific task or screen the user is currently working on.

g) A _____ interface allows users to interact with different parts of an application without being restricted to a particular mode.

h) In a modeless interface, users can perform _____ tasks simultaneously within the same application.

i) A common example of a modeless interface is _____ Word.

j) A _____ User Interface allows users to interact with a system using graphical elements like icons and menus.

k) A Text-Based User Interface mainly uses _____ for interaction.

l) GUI requires more _____ and processing power compared to a text-based interface.

Q5. Answer the following questions:

a) What is meant by 'user-cantered design' in UI design?

b) Why is consistency important in user interface design?

c) Explain the concept of feedback with an example.

d) What is user guidance?

e) Mention any two features of user guidance.

f) What is online help? Explain any one type of online help.

g) What is a modeless interface?

h) Mention any two features of a modeless interface.

i) State one advantage and one disadvantage of a modeless interface.

j) What is a Graphical User Interface (GUI)?

k) State any two differences between a GUI and text-based user interface (TUI).

l) Mention any two advantages of a text-based interface.

10.5 FUNDAMENTALS OF COMPONENT-BASED GUI DEVELOPMENT

Component-based GUI development is a method of designing graphical user interfaces by using reusable and independent components such as buttons, text fields, labels, menus, and panels. Each component performs a specific function and can be combined with others to create a complete application interface.

1) Concept of Components: A component is a reusable software element that represents a graphical control or object in a user interface. Each component has its own properties, methods, and events.

***Examples of GUI components*:**

- Buttons
- Text fields
- Labels
- Check boxes
- Radio buttons
- Menus
- Dialogue boxes

These components can be arranged on a window or form to create the user interface.

2) Properties of Components: Properties define the appearance and behavior of a component.

***Examples include*:**

- Size and position
- Color and font
- Text displayed on the component
- Visibility and enable/disable state

Developers can modify these properties to customize the interface.

3) Events: An event is an action that takes place when a user interacts with a component.

***Examples*:**

- Clicking a button.
- Pressing a key.
- Selecting a menu item.
- Moving the mouse

The system responds to these actions through event handling.

4) Event Handling: Event handling is the mechanism that defines how the system responds when an event occurs.

***Example*:**

- When a user clicks a Submit button, the system processes the entered data.
- When a user selects a menu option, the corresponding function is executed.

5) Layout Management: Layout management refers to arranging GUI components properly within a window or form. Common layout methods include:

- Grid layout
- Flow layout
- Border layout

Layout managers help maintain proper alignment and spacing of components.

6) Reusability of Components: Components can be reused in multiple applications. This reduces development time and improves software consistency.

***Example*:** A login form component can be reused in many different applications.

Advantages of Component-Based GUI Development

- Faster application development
- Reusable components
- Easier maintenance
- Improved modularity
- Consistent user interface design

CHECK YOUR PROGRESS

Q6. Fill in the blanks:

a) Component-based GUI development uses _____ and independent components to build user interfaces.

b) A component has its own properties, methods, and _____.

c) _____ define the appearance and behavior of a component.

d) An _____ is an action that occurs when a user interacts with a component.

e) _____ management helps in arranging components properly within a window or form.

Q7. Answer the following questions:

a) What is component-based GUI development?

b) What is meant by a GUI component? Give two examples.

c) What are properties of components? Provide examples.

d) What is event handling in GUI development?

e) What is layout management? Name any two layout methods.

10.6 A USER INTERFACE DESIGN METHODOLOGY

A user interface (UI) design methodology is a systematic process used to design effective and user-friendly interfaces. It focuses on understanding user needs, designing the interface, testing it, and improving it to ensure better usability.

The main steps involved in a UI design methodology are described below.

1) **User Analysis:** The first step is to understand the users of the system. Designers study the users' needs, skills, experience, and expectations.

Activities involved:

- Identifying target users
- Studying user behavior and requirements
- Understanding the environment in which the system will be used

This step ensures that the interface is designed according to the users' capabilities.

2) **Task Analysis:** In this stage, designers analyze the tasks that users need to perform with the system.

Objectives:

- Identify the sequence of tasks
- Understand how users interact with the system
- Determine the information required to complete tasks

Task analysis helps in designing interfaces that support efficient task completion.

3) **Interface Design:** Based on the user and task analysis, designers create the layout and structure of the interface.

Elements considered:

- Screen layout
- Menus and navigation
- Icons and buttons
- Input and output formats

The goal is to create an interface that is simple, consistent, and easy to use.

4) **Prototyping:** A prototype is an early model or sample of the interface.

Purpose of prototyping:

- Demonstrate how the interface will look and work
- Allow users to test the design before final development
- Identify design problems early

Prototypes may be low-fidelity (sketches) or high-fidelity (interactive models).

5) **Evaluation and Testing:** In this step, the interface is tested with real users to evaluate its usability.

Methods include:

- Usability testing
- User feedback
- Observation of user interaction

This helps identify errors, difficulties, and areas for improvement.

6) **Implementation and Refinement:** After testing, the final interface is implemented in the system. Designers may continue improving the interface based on feedback and performance.

CHECK YOUR PROGRESS

Q8. Fill in the blanks:

a) A UI design methodology is a _____ process used to design effective interfaces.
b) _____ analysis focuses on understanding the users' needs, skills, and expectations.
c) Task analysis helps in designing interfaces for efficient _____ completion.
d) A _____ is an early model or sample of the interface used for testing design ideas.
e) _____ testing involves evaluating the interface by observing real users.

Q9. Answer the following questions:

a) What is a user interface design methodology?
b) What is the purpose of user analysis in UI design?
c) What is task analysis? Why is it important?
d) What is prototyping? Mention its purpose.
e) What happens during the evaluation and testing stage of UI design?

10.7 LET US SUM UP

- A User Interface (UI) is the point at which a user interacts with a computer system, software application, or digital device.
- User Interface Design is the process of creating the visual layout and interactive elements of a system to ensure that users can interact with it in an effective, efficient, and satisfying way
- The main goal of UI design is to make the interaction between the user and the system simple, efficient, and user-friendly.
- User Guidance and Online Help are essential components of a software system that support users in understanding and using the application effectively.
- User Guidance refers to the support provided to help users understand how to

operate a system or application.

- A modeless interface is a type of user interface that allows the user to interact with different parts of the application without being restricted to a particular mode
- Two common types of interfaces are the Graphical User Interface (GUI) and the Text-Based User Interface (TUI).
- Component-based GUI development is a method of designing graphical user interfaces by using reusable and independent components such as buttons, text fields, labels, menus, and panels.
- A user interface (UI) design methodology is a systematic process used to design effective and user-friendly interfaces. It focuses on understanding user needs, designing the interface, testing it, and improving it to ensure better usability.

✓ 10.8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1:

a) ii) Consistency,

b) iii) User needs and experience,

c) ii) Consistency,

d) iii) Keeps users informed about system status,

e) ii) Learnability

Answer of Q2:

a) understand

b) consistency

c) feedback

d) efficiency

e) accessibility

Answer of Q3:

The two characteristics of a good user interface with brief explanations are:

1. **Simplicity:** A good user interface should be simple and easy to understand. It should avoid unnecessary elements so that users can perform tasks quickly without confusion.
2. **Consistency:** Consistency ensures that similar elements and actions behave the same way throughout the system. This helps users learn the interface easily

and reduces errors.

Answer of Q4:

a) user-centered

b) usability

c) simplicity

d) help

e) tooltips

f) context-sensitive

g) modeless

h) multiple

i) Microsoft

j) graphical

k) text commands

l) memory

Answer to Q5:

a) User-centred design is an approach that focuses on designing interfaces based on the needs, preferences, and abilities of users to ensure ease of use and satisfaction.

b) Consistency helps users learn and use the system easily because similar actions produce similar results throughout the interface, reducing confusion and errors.

c) Feedback is the system's response to a user's action. For an example, when a user saves a file, a confirmation message like "File saved successfully" is displayed.

d) User guidance refers to the assistance provided to help users understand how to operate a system or application effectively.

e) The two features of user guidance are – Tooltips and Help messages.

f) Online help is a built-in support system within software that provides instructions and information to users. For an example, Context-sensitive help gives information related to the current task or screen.

g) A modeless interface allows users to interact with different parts of an application without being restricted to a single mode.

h) The two features of a modeless interface are –

- Allows multiple tasks to be performed simultaneously
- Provides greater flexibility in user interaction

i) Here one advantage and one disadvantage of a modeless interface is stated below:
   - Advantage: Users can work on multiple tasks at the same time
   - Disadvantage: It may confuse beginners due to multiple active options

j) A GUI is a type of user interface that allows users to interact with a system using graphical elements such as icons, menus, windows, and buttons.

k) Difference between GUI and TUI are given below:
   - GUI uses graphical elements, whereas TUI uses text commands
   - GUI is user-friendly, whereas TUI requires command knowledge

l) The advantages of a text-based interface are –
   - Requires less memory and processing power
   - Faster for experienced users

Answer to Q6:

a) Reusable

b) Events

c) Properties

d) Event

e) Layout

Answer to Q7:

a) Component-based GUI development is an approach to designing user interfaces by using small, reusable, and independent building blocks called components. Each component performs a specific function and can be combined with others to create a complete interface. This approach improves efficiency, consistency, and maintainability.

b) A GUI component is an individual element of a graphical user interface that allows users to interact with a system. Each component has its own functionality and appearance. For example, Button, Text Box, etc.

c) Properties are the attributes that define the appearance and behaviour of a component. They control how a component looks and how it functions. For example:
   i. Colour (background or text colour)
   ii. Size (width and height)
   iii. Font style

d) Event handling is the process of responding to user actions such as clicking a button, typing on the keyboard, or moving the mouse. When an event occurs, the

system executes a specific set of instructions (called an event handler) to perform the required task.

e) Layout management is the process of arranging and organizing GUI components within a window or form in a structured way. It ensures that components are placed properly and adjust correctly when the window size changes.

Two layout methods are:

i. Flow Layout

ii. Grid Layout

Answer to Q8:

a) Systematic

b) User

c) Task

d) Prototype

e) Usability

Answer to Q9:

a) A user interface design methodology is a structured and systematic process used to design effective, user-friendly interfaces. It involves a series of steps such as analysis, design, prototyping, and evaluation to ensure that the interface meets user needs and provides a good user experience.

b) The purpose of user analysis is to understand the target users, including their needs, preferences, skills, and limitations. This helps designers create interfaces that are easy to use, relevant, and suitable for the intended audience.

c) Task analysis is the process of identifying and studying the tasks that users need to perform using the system. It is important because it helps designers organize features and workflows in a way that allows users to complete tasks efficiently and accurately.

d) Prototyping is the process of creating an early model or sample of the user interface. The purpose of a prototype is to visualize design ideas, test functionality, gather feedback from users, and identify problems before developing the final system.

e) During the evaluation and testing stage, the interface is tested with real users to assess its usability, efficiency, and effectiveness. Designers observe user interactions, identify issues or errors, and collect feedback to improve the design before final implementation.

10.9 FURTHER READING

- Shneiderman, B. et al., 6th edition (2022). A classic textbook covering fundamental principles, guidelines, and best practices in UI design. Pearson/ Addison-Wesley.
- Lauesen, S. (2005). User Interface Design: A Software Engineering Perspective. Pearson/Addison-Wesley.
- Benyon, D. 5th edition (2019). Designing Interactive Systems. Pearson Education.
- Mall, R. 5th edition (2018). Fundamentals of Software Engineering. PHI Learning Pvt. Ltd.
- Aggarwal, K.K. and Singh, Y. (2000). Software Engineering (UI Design Chapter). New Age International Publishers.

10.10 MODEL QUESTIONS

Q1. Explain accessibility and error prevention and recovery in UI design with suitable examples.
Q2. Compare user guidance and online help with suitable examples.
Q3. Explain how user guidance and online help improve user experience and system usability.
Q4. Discuss the features, advantages, and disadvantages of a modeless interface.
Q5. Compare a modeless interface with a modal interface (conceptual comparison).
Q6. Compare Graphical User Interface (GUI) and Text-Based User Interface (TUI) in detail.
Q7. Discuss different types of user interfaces with suitable examples.
Q8. Explain Natural Language Interface and Touch-Based Interface with their features and examples.
Q9. Compare different types of user interfaces based on usability, interaction method, and applications.
Q10. Explain the fundamentals of component-based GUI development.
Q11. Discuss the concept of components, events, and event handling in GUI development.
Q12. Explain the advantages of component-based GUI development with suitable examples.
Q13. Discuss the steps involved in a user interface design methodology.
Q14. Explain user analysis and task analysis with suitable examples.
Q15. Describe the importance of evaluation, testing, and refinement in UI design.

****

UNIT 11: SOFTWARE CODING

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Coding

11.4 Coding Standards and Guidelines

11.4.1 Common Coding Standards

11.5 Code Review

11.6 Code Inspection

11.7 Code Walkthrough

11.8 Cleanroom Testing

11.9 Software Documentation

11.9.1 Internal Documentation

11.9.2 External Documentation

11.10 Check Your Progress

11.11 Let Us Sum Up

11.12 Answers to Check Your Progress

11.13 Further Reading

11.14 Model Questions

11.1 LEARNING OBJECTIVES

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

- understand the concept of coding and importance of coding standards and guidelines in software development.
- explain the code review, code inspection, code walkthrough and cleanroom testing process and their importance in developing quality software.
- identify and differentiate between internal and external software documentation and understand their importance in software development and maintenance.

11.2 INTRODUCTION

In the previous unit, we learned about user interface design. In this unit we will learn about coding standards and guidelines. We will discuss about the different coding verification

techniques such as code review, code inspection, code walkthrough and cleanroom testing. These are static code verification techniques which ensure that the software code adheres to the system and design specifications and meets user requirements. At the end of the unit, we will also discuss the importance of software documentation and different types of software documentation. In the next unit, we will explore the concept and different types of software testing.

11.3 INTRODUCTION TO SOFTWARE CODING

Coding is one of the key phases in the Software Development Life Cycle (SDLC). In this phase, the developers write instructions in a systematic order using different languages like C, C++, Java, Python, and others, to direct the computer what to do. In this phase, the designs, algorithms, and the requirements from the previous phases are translated into executable code. Coding involves logical and systematic ordering of the instructions written using programming languages to solve a specific task efficiently. Coding enables automating tasks through the development of software and applications.

The following are the most common building blocks of a code -

- ***Variables:*** Variables are used to store information such as numbers, text, lists, dictionaries, etc. Each programming language has its own set of rules for naming variables.
- ***Instructions / Statements:*** These are the lines which direct the computer to perform some logical or arithmetic operations. In most programming languages, instructions are executed sequentially one after another. However, there are some specific instructions like conditional statements, function calls, loops, etc. that may alter the flow of execution.
- ***Operators:*** Operators are used to perform arithmetic operations (like +, -, /, % etc.) or logical/comparison operations (like AND, OR, >, <, == etc.).
- ***Functions:*** These are the named blocks of instructions that can be reused.
- ***Syntax and Punctuation:*** Each programming language has a set of rules (grammar) to be followed while writing the instructions. For example, use of semicolon (;), colons (:), brackets ({ }), indentation are to be followed properly to make the code valid in a particular language.

11.4 CODING STANDARDS AND GUIDELINES

All the good software development organisations maintain a set of guidelines to be followed by the developers while coding. These sets of guidelines are referred to as coding standards. Coding standards often cover many aspects such as naming conventions, code formatting, usage of different types of variables, lengths of instructions, error handling,

and many more. These guidelines help developers to create uniform, consistent, and readable co. Organisations often set their own coding standards based on the types of software products they deal with.

Advantages of Using Coding Standards and Guidelines

The following are some benefits of having coding standards-

- In a big project multiple developers work together as a team. The work is divided among the team members. Adherence to coding standards helps them to write uniform and consistent code instead of everyone following different styles.
- It adds to seamless integration of code developed by different programmers.
- Coding standards make the code more readable and maintainable.
- It reduces the complexity of the code and improve the efficiency of the programmers leading to faster development.
- It increases the reusability of programs and make detection of bugs easier.
- It makes the code review process easier.
- Many guidelines prevent common vulnerabilities and thus ake the code more secure and protected.

11.4.1 Common Coding Standards

The following are some common coding standards followed in good software companies-

- **Limited use of global variables:** Minimise the use of global variables to reduce side effects like unintended modifications. Limited use of global variables also increases the code modularity and helps in code testing.
- **Avoid goto statements:** Goto statements make the code unstructured and make the program difficult to trace.
- **Standard Headers:** This rule states that a standard header should be maintained for each module/file. This adds to better understanding, maintenance, and traceability of the source code modules/files. The header section must include some basic information like name of the module, date of creation, author of the module, modification history, module summary, list of global variables used/ modified, etc.
- **Use of comments:** Comments are the non-executable statements included in the code for explaining what a particular section of the code does. It is considered one of the good practices to increase the readability and traceability of the code. However, in modern days the code is emphasised to be self-explanatory and thus comments are included only to explain the complex parts of the code.

- **Naming conventions:** Names of the identifiers must be meaningful and self-explanatory. Naming should also be consistent for different types of identifiers. Some of the common naming conventions are-
    - Use meaningful names. Eg., userAge instead of ua, u, etc.
    - Use camelCase for local variables (e.g., localVariable, surfaceArea).
    - The names of global variables should start with capital letters (e.g., Global, Radius).
    - Use uppercase for constants. E.g., PI, CONST_DATA.
    - For functions and methods, names should be in camelCase. E.g., calculateTotalArea().
    - Digits should be avoided while naming the variables.
    - An identifier should not be used for multiple purposes. It makes the understanding of the code difficult and may lead to confusion at the time of development.
- **Indentation:** While writing the instructions, the indentation should be constant throughout the code (typically 2 or 4 spaces). Proper indentation improves the readability of the code and makes it easier to locate syntax errors. Each nested block must be indented consistently. A logical block must also be preceded and followed by proper indentation.
- **Length of instructions and functions:** Instructions should not be too long. The number of characters in an instruction should be limited to 80–120. Similarly, functions also should not be too lengthy (ideally < 50–100 lines). Bigger tasks should be divided into smaller tasks to increase the modularity of the program. This also leads to smaller functions which are easier to understand and debug.
- **Error handling:** Error reporting should be consistent throughout the code (e.g., 0 for success, 1 or negative value for failure). This makes the debugging process easier. Also, the error handling blocks such as try/catch should be used wherever applicable. These blocks should return meaningful messages to the user whenever any error/exception is encountered.

11.5 CODE REVIEW

In the software development life cycle, to ensure quality, the source code is examined thoroughly by peers before it is passed on to the testing and deployment phase. In this phase, static verification is carried out, where the code is analysed systematically without executing it. Code review, code inspection, and code walkthrough are the three popular techniques for carrying out static verification.

Code review is a widely used static verification technique where, once a developer completes the code, it is reviewed systematically by peers. While reviewing, they mostly look into the following areas:

- Does the code meet the system requirements?
- Does the code adhere to the designs?
- Does the code adhere to the coding standards?
- Is the code legible?
- Are there any potential vulnerabilities in the code?
- Is there any logical flaw in the code?

After critically scrutinising the code for these aspects, the reviewers submit their feedback. Based on the comments, the developer makes and submits the changes. The reviewers add comments, the developer responds and revises. This cycle goes on until the code is fully approved by the reviewers. In modern days, the code review process is supported by many automated tools such as linters and formatters. These tools help to identify logical and formatting discrepancies fast and thus have made the reviewing process much more efficient.

Advantages of code review:

- **Improved code quality:** Code review ensures that the coding standards are followed and the code is well formatted and well documented. Also, in this phase, potential security vulnerabilities, logical errors, and other performance issues are identified. This leads to fewer defects going to the testing phase.
- **Knowledge sharing and learning:** The reviewers and the developers learn from each other during the review process. The reviewers share their knowledge with the developers and give valuable feedback, while the developers learn from the suggestions and get exposure to best practices.
- **Enhanced team collaboration:** The reviewing process involves discussion, positive feedback, and collective ownership. This helps to build trust among the team members and turns individual work into shared responsibilities.
- **Long-term cost savings:** Although the review process takes some time, it helps to save time during the later testing phase.

Disadvantages of code review:

- Code review often takes a significant amount of time. The review process itself takes some time. Waiting for the review comments, discussing the comments, and making the suggested changes often cause delays to the later phases.
- Developers get engaged in the reviewing process rather than writing new code. In

small teams, this may become an overhead.

- When there are too many review requests and there are a limited number of experienced reviewers, the review quality may go down. Inexperienced reviewers may miss some crucial points, reducing effectiveness.
- Absence of clear guidelines leads to an ineffective review process. complex calculations, manage data, and perform a wide range of tasks quickly and accurately.

11.6 CODE INSPECTION

Coming inspection, also known as Fagan inspection, is a more formal process. Developed by Michael Fagan at IBM in the 1970s, it is a highly structured process designed to enhance the efficiency of defect detection. It is a rigorous process which involves five to eight persons with the following key roles-

- **Author**, who writes the code
- **Moderator,** who leads the process
- **Reader**, who reads the code aloud line by line
- **Inspector,** who looks for defects. There may be more than one inspector.
- **Recorder,** who logs the identified issues.

Steps of code inspection

- ***Planning:*** The code inspection process is led by and planned by the moderator. The relevant materials are given to the inspection team and then a meeting is scheduled.
- ***Overview meeting:*** In this meeting, the author gives a brief overview and summary of their code to the inspection team.
- ***Preparation:*** In this stage, each member in the inspection team individually reviews the code using a predefined checklist.
- ***Inspection meeting:*** In this meeting, the reader reads out the code aloud line by line and the inspectors point out the flaws. The recorder keeps a record of the identified issues. The focus is completely on pointing out the defects rather than fixing them. The defects are also classified into different severity classes.
- ***Rework:*** The author modifies the code to eliminate the defects.
- **Follow up:** A meeting is held with the team to discuss the reviewed code.

Advantages of code inspection:

- Though time- and resource-intensive, formal code inspection has been proved to be effective in finding the defects in the early stage.

- This process improves the overall quality of the code.
- The author learns from the feedback provided by the inspectors.

Disadvantages of code inspection:

- The process is time-consuming and resource-heavy.

11.7 CODE WALKTROUGH

Unlike code inspection, code walkthrough is an informal process. The authors initiate a meeting with the team members, where they present and explain the code. The team members are usually provided with the code and the documents beforehand. The members discuss their findings with the authors and give their suggestions. In the walkthrough sessions, the members not only point out the issues, but also discuss the solutions. Thus, walkthroughs are informal but purposeful sessions.

Advantages:

- It is less time consuming and does not require extensive preparation.
- The author gets a variety of viewpoints.
- Each team member gets an insight into the others' code, they can discuss complex algorithms, discuss early prototypes. This helps the freshers to understand the code.

Disadvantages:

- If the team members are in different zones, arranging meetings can be difficult. The meetings are sometimes also time consuming.
- The code walkthrough process, as it is very informal, may sometimes lack depth and some issues may remain unidentified.

11.8 CLEANROOM TESTING

Cleanroom testing is a preventive approach which uses mathematical formulation, rigorous verification and certification for reliability. The name is derived from the semiconductor manufacturing cleanrooms where manufacturing is carried out in ultra clean rooms to prevent any defects. Cleanroom testing emphasizes zero-defect software.

Principles of Cleanroom Testing:

- Prevention is emphasized over detection. The primary focus is to identify and eliminate errors without relying on the testing phase.
- Software specifications and designs are mathematically specified. Quantitative analysis of the code is done using statistical methods and metrics. This adds to data driven decision making leading to continuous improvement.

- Incremental development policy is followed; here the system is divided into small and certifiable increments.
- Code is verified using mathematical proofs and correctness. Both verification and validation are given utmost importance. Validation ensures that users' requirements and specifications are met while verification ensures that the product fits the stated criteria.
- End users also participate in cleanroom testing. Thus they can check if their requirements are met.
- Developers submit code for independent verification and certification. Random test cases are generated from use case models and the test results are analysed to provide certified reliability estimates.

Advantages of cleanroom testing:

- Cleanroom testing results in highly reliable code. Early and systematic detection and fixing of errors reduces future rework.
- Quantifiable and certifiable statistics aid data driven decision making. This boosts productivity and reliability and thus is used in high stake domains such as aerospace, defence, medical devices and many more.

Disadvantages of cleanroom testing:

- This approach is resource intensive and time consuming. Mathematical and statistical experts are needed for mathematical translation of the user and design specifications.
- Not suitable for non-critical projects like simple websites or mobile applications.
- Statistical testing may not catch some rare edge cases if usage models are incomplete.
- Not ideal for rapidly changing requirements.

11.9 SOFTWARE DOCUMENTATION

Software documentation contains the description of the software, its design, functionality, development and usage. It serves as the link between the users, developers and stakeholders which contains information about how the software works, how it should be installed, and how it should be maintained. These documents also ensure that knowledge is preserved even if the development team changes.

Software documentation can broadly be classified into

- Internal Documentation
- External Documentation

11.9.1 Internal Documentation

Internal documentation is maintained mainly for the developers and development team. In these documentation technical details are maintained, which helps the developers to understand the design, structures and working of the components. It thus makes the modification, debugging and maintenance process easier. Internal documentation also improves the readability and quality of the code and assists the developers to understand the code easily. There are different types of internal documentation.

Following are some of the important internal documentations–

- **Comments in the source code**: Comments are added inside the code to explain complex logic or purpose of a block.
- **Meaningful function and variable names:** By using meaningful names for the variables and functions the code becomes much easier to understand.
- **Program structure documentation:** This explains the different structural components of the software like modules, functions, classes, relationship among the components and how they interact.
- **Algorithm, Flowchart and Pseudocode:** Algorithm is a sequence of logical steps to solve a particular problem. Flowchart is the visual representation of the algorithm, depicting the flow of control from one step to another. Pseudocode, on the other hand, is a simplified representation of the program logic using a mixture of natural language and programming-like statements.
- **System Designs:** This includes system architecture used, list of data structures used, and different algorithms that have been used to develop the software.

11.9.2 External Documentation

External documentation are mainly prepared for the end users, system administrators and other non-technical stakeholders outside the development team. It explains how to install, operate and maintain the software. External documentation also guides the user to resolve issues that may arise while using the software by providing troubleshooting tips. These documents are usually provided as user manuals, guides or help files.

External documentations may be in different forms. Some of those are –

- **User Manual:** This document guides the user on how to use the software. It may include instructions for operating the system, screenshots, examples and troubleshooting tips. For example, a user manual may contain guidelines on how to create an account, how to enter data and how to generate reports.
- **Installation Guide:** This document includes the steps on how to install the software. It may contain information about system requirements, installation steps,

configuration instructions and troubleshooting during installation.

- **System Documentation:** This document has information on the overall structure of the software system including system architecture, data flow diagrams, software and hardware requirements, and database designs.
- **Operational Documentation:** This document is for the system operators which includes information on system operation, backup procedures and security measures.
- **Troubleshooting Guides:** This document helps the users to identify and resolve the common problems that may arise while installing and operating the software.

11.10 CHECK YOUR PROGRESS

Q1. What is software coding and why is it considered as an important phase in the Software Development Life Cycle (SDLC)?

Q2. What are coding standards and guidelines?

Q3. What is code review? In a code review process, what are the common aspects that are looked into?

Q4. Define code inspection. What are the different roles of the participants involved in code inspection process?

Q5. What is a code walkthrough? How does it differ from code inspection?

Q6. What is cleanroom testing?

Q7. What is software documentation? Why is it important?

Q8. Differentiate between internal documentation and external documentation.

Q9. Fill in the blanks:

i. Coding is one of the important phases in the _____.

ii. _____ are non-executable statements added in the code to explain complex logic.

iii. Code inspection is also known as _____ inspection.

iv. Cleanroom testing focuses on developing _____ defect software.

11.11 LET US SUM UP

- In the Software Development Life Cycle (SDLC), coding is the phase where the requirements and designs are translated into instructions. This phase results in executable code.
- Coding can be done using different languages like C, C++, Java and Python.
- Coding standards help the developers to develop uniform, readable and maintainable code. Adherence to coding standards also makes the code less

complex and enhances the quality of the developed code.

- Static verification techniques such as code review, code inspection, code walkthrough and cleanroom testing are employed to analyse the code systematically to identify defects prior to the testing phase.
- Code review is done by peers, who check if the code satisfies various aspects such as whether the code meets the coding guidelines, whether there are any security vulnerabilities or logical errors, and whether the code meets the system specifications.
- Code inspection is a more formal way to verify the code. In this process, the participants are divided into different roles such as moderator, author, reader, recorder and inspector. Though it is time consuming, it leads to more reliable code.
- Code walkthrough, on the other hand, is an informal process where the focus is on identifying and fixing bugs. The process is led by the author of the code who explains and discusses the code with the team members, who in turn give their feedback on potential issues.
- Cleanroom testing is an error prevention approach with the goal of producing code with zero defects. It is a rigorous process based on mathematical formulation and quantitative evaluation.
- Software documentation contains the description of the system, its components and their functionalities, design diagrams, installation guides etc. Software documentation can be both internal and external. Internal documentation is maintained for the developers while external documentation is made for the end users.

✓ 11.12 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: Software coding is the process of translating the design and system specifications into programming statements. Code tells the system exactly what is to be done and thus performs some specific task. This phase results in executable code.

Answer to Q2: Coding standards are a set of guidelines that are to be followed by the developers while developing the code. These are important because these guidelines help the developers to develop uniform and unambiguous code. Adherence to the coding standards makes sure that the code is readable, maintainable and simple. It also enhances the quality of the code.

Answer to Q3: Code review is a static code verification technique. In this process, the code is evaluated by peers, who look for certain aspects like compliance with the coding standards and system specifications, presence of any logical error, and potential security

issues. Based on the comments, the author of the code updates the code.

Answer to Q4: Code inspection is a formal and structured process to analyse the code to detect the presence of any defects. It is carried out before the testing phase. The participants in the code inspection process are divided into different roles, which are the author, moderator, reader, inspectors, and recorder.

Answer to Q5: Code walkthrough is an informal process where the author initiates a meeting and explains the code to the team members. The team members give their viewpoints and suggestions on the developed code.

Unlike code inspection, which is a formal structured process, code walkthrough is an informal process. Code inspection is time and resource intensive, whereas code walkthrough does not require a lot of advanced preparation. As code walkthrough is done within the team, it is also less resource intensive. Moreover, in code inspection the focus is on identifying the defects, while in code walkthrough the focus is on correcting the defects.

Answer to Q6: Cleanroom testing is a preventive technique that focuses on developing zero-defect software. It uses mathematical specifications, formal verification and quantitative metrics to evaluate the developed code. It is a rigorous process which uses quantifiable and certifiable statistics.

Answer to Q7: Software documentation includes the description of the software, its components and their functions, design specifications, installation guidelines and debugging tips. It is very useful for maintenance of the software. It ensures that the knowledge is preserved even if the developer leaves the organisation or is no longer associated with the project. It also helps the end users and system operators by providing the basic knowledge of how to install the software and fix common bugs.

Answer to Q8: Internal documentation is for the developers which usually contains information like system requirements, design specifications, algorithms, flowcharts and pseudocode. It also contains explanations of different components used in the program like functions, modules, classes and relationships.

External documentation, on the other hand, is meant for the end users and comes in the form of user manuals, guides and help documents. These documents contain information like how to install the software and how to troubleshoot some common errors while operating the software.

Answer to Q9: i) Software Development Life Cycle

ii) Comments

iii) Fagan

iv) Zero

11.13 FURTHER READING

- Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.
- Mall, R. (2014). Fundamentals of Software Engineering (4th ed.). PHI Learning.
- Pressman, R. S., & Maxim, B. R. (2015). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill Education.
- https://www.geeksforgeeks.org/software-engineering/coding-standards-and-guidelines/
- https://blog.codacy.com/coding-standards
- https://www.geeksforgeeks.org/software-engineering/code-review-in-software-engineering/
- https://www.tutorialspoint.com/software_engineering

11.14 MODEL QUESTIONS

Q1. Discuss some common coding standards followed by good software companies. Why are coding standards important?

Q2. Explain the common building blocks of a code.

Q3. State some important naming conventions. What naming conventions are important?

Q4. Discuss the code review process. What are its advantages and disadvantages?

Q5. Explain the different roles and steps involved in the code inspection process. Mention its advantages and disadvantages.

Q6. What is code walkthrough? Explain how code walkthrough sessions contribute to knowledge sharing and team collaboration in software projects.

Q7. Discuss how the code review, code inspection and code walkthrough processes help in developing quality code.

Q8. Discuss the roles and responsibilities of the participants in the code inspection process.

Q9. Discuss the principles of cleanroom testing.

Q10. Discuss the limitations of cleanroom testing and explain why it may not be suitable for all software projects.

Q11. Explain the importance of software documentation in the maintenance phase of the software life cycle.

Q12. Explain the objectives of internal and external documentation. Also discuss their content and different forms.

****

UNIT 12: SOFTWARE TESTING

UNIT STRUCTURE

12.1 Learning Objectives
13.2 Introduction
12.3 Software Testing
12.4 Testing Strategies
12.5 Types of Testing
12.5.1 Black-Box Testing
12.5.2 White-Box Testing
12.5.3 Stress Testing
12.5.4 Storage Testing
12.5.5 Performance Testing
12.5.6 Unit Testing
12.5.7 Integrating Testing
12.6 Let Us Sum Up
12.7 Further Readings
12.8 Answers To Check Your Progress
12.9 Model Questions

12.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- define the term Software Testing
- explain the importance of testing process
- describe their characteristics
- identify the basic testing types
- explain the testing strategies
- distinguish between different testing methods
- identify testing data

12.2 INTRODUCTION

Software Testing is the last stage of software development life cycle. We have already discussed it in brief in the previous units. Now, in this unit, we are going to have a detailed discussion on it, its different types and levels.

In developing software, errors may occur at any stage and it may be detectes after each phases. But the final code is likely to have some requirement errors and design

errors which have to be checked at the end. Testing is the activity where errors remaining from all the previous phases must be detected. Hence, testing performs a critical role for ensuring quality. This unit focuses on the entire system testing with different levels. It is important to choose test data for testing. This is also being discusses here. In the next unit, we will explore the concepts related to software maintenance and quality control.

12.3 SOFTWARE TESTING

Software testing is a critical element of software quality assurance and it represents the ultimate review of specification, design and coding. Testing is generally done in two levels - testing of individual modules and testing of entire system called *system testing*. During system testing, the system is used experimentally to ensure that the software does not fail. It should run according to its specifications and in the way the users expect. A limited number of users are allowed for testing. Analyst tries to discover any difficulties of the system before the organization implements the system. It is a very tedious and time consuming job. It does not guarantee the reliability of the system, but can assure the system to be error free.

Why do we do testing? Because system testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved. Effective testing translates directly into long-term cost saving from a reduced number of errors. Another reason is its *utility* as a user-oriented vehicle before implementation. Unfortunately, the user's demands are often compromised by efforts to facilitate program or design efficiency in terms of processing time or memory utilization.

Testing is the only step in the software engineering process that can be viewed as destructive rather than constructive. The engineer creates a series of test cases that are intended to demolish the software that has been built. However, the objectives of testing are somewhat different from what they are expected to be. A set of rules for testing objectives given by Glen Myers are as follows:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.
  From this, it is understood that a successful test is one in which no errors are found.

12.4 TESTING STRATEGIES

Testing basically is a verification and validation process to make sure that the system is satisfying the conditions and the requirements strategies imposed at the time of starting the project. There are two general strategies of testing software- code testing and specification testing.

Code testing: This strategy examines the logic of the program. In this method an analyst develops test cases that result in executing every instruction in the program or module. Here every **path** through the program is tested.

Specification testing: For specification testing, an analyst examines program specifications that state what the program should do and how it should perform under various conditions. Then, test cases are developed for each condition or combination of conditions and submitted for processing. It looks at the program as a whole. The assumption here is that if the program meets the specifications it will not fail.

CHECK YOUR PROGRESS

Q1. Fill in the blanks:

a) Testing is a very tedious and _____ job.

b) Each _____ is designed with the intent of finding errors in the way the system is process it.

c) _____ strategy examines the logic of the program.

d) _____ test data are actual data extracted from file.

Q2. Write True or False:

a) In specification testing analyst examines the program specification.

b) In code testing analyst examines the whole program structure.

c) Testing is done only at the end of the system development process.

d) Artificial test data are solely for test purposes. Artificial test data are generated by programmer itself

12.5 TYPES OF TESTING

Testing is done in different stages of software development process and in different ways and accordingly it is categorized. There are basically two types of testing approaches: the **black-box testing** and **white-box testing**. Apart from these two approaches there are many more which are being discussed in this unit. But these also fall directly or indirectly in these two.

12.5.1 Black-Box Testing

In black-box testing the structure of the product is not considered. Test cases solely depend on the requirement specification of the program or module. If the product function is known, this test is conducted. This testing is also called *functional or behavioral testing*. It is most impractical and exhaustive testing. This test is not responsible for internal structure of functions. One criterion for generating test cases is to generate them randomly. There are no formal rules for designing test cases for functional testing. In fact, there are no precise criteria for selecting test cases. However, there are some techniques that can be used to select test cases that have been found to be very successful in detecting errors. However these techniques are not discussed here and are left for further reading.

12.5.2 White-Box Testing

If the internal workings of a product are known, white-box tests can be conducted to assure that internal operation performs according to specification. We have seen in the previous section that black-box testing is concerned with functionality rather than implementation of the program. White-box testing on the other hand is concerned with testing the implementation of the program. The intent of this testing is not to exercise all the different input or output conditions, but to exercise the different programming structure and data structures used in the program. It is also called *structural testing*. As this test is based on formal structure, the criterion for this test is formal and precise. It aims to achieve test cases that will force the desired coverage of different structures.

12.5.3 Stress Testing

The purpose of stress testing is to prove that the candidate system does not malfunction under peak loads. It depends on time factor also. This simulates an online environment where a high volume of activities occurs in spurts. This testing is used to evaluate how the system will work in unavoidable condition also. This is one part of black-box testing.

12.5.4 Storage Testing

This test is to be carried out to determine the capacity of the system to store transaction data on a disk or in other files. Capacities here are measured in terms of the number of records that a disk will handle or a file can contain. If this test is not carried out then there are possibilities that during installation one may

discover that, there is not enough storage capacity for transactions and master file records.

12.5.5 Performance Testing

This test refers to the response time of the system being installed. It is conducted prior to implementation to determine how long it takes to receive a response to inquiry, make a back up copy of a file, or send a transaction and receive a response. This also includes test runs to time indexing or resorting of large files of size to prepare a report. A system may run well with only a handful of test transactions may be unacceptably slow when fully loaded. This should be done using the entire volume of live data. By this testing, the speed and the effectiveness of the system is determined. It is also seen whether the system is performing as per the requirement of the client. It falls under the category of black-box testing.

12.5.6 Unit Testing

It focuses on each individual module, assuring its functions properly as a unit. This is also called program testing. Multiple modules can be tested parallelly and these are assembled and integrated at last to perform a specific function. Five primary characteristics of a module are evaluated during unit testing:

- Module interface
- Data structure
- Important execution paths
- Error-handling paths
- Boundary conditions

Tests of data flow across a module interface are required before any other test is initiated. If data do not enter and exit properly, all other tests may fail. Selective testing of execution paths is an essential task during the unit test.

Normally, unit testing is considered an adjunct to the coding step. After source level code has been developed, reviewed and verified for correct syntax, unit test case design begins. Because a module is not a stand-alone program, the driver, the main program and the routine software must be developed for each unit test. A testing sub program uses the sub routine module's interface; it may do minimal data manipulation, print verification of entry, and returns control to the super ordinate module. Unit testing is simplified when a module with a high degree of cohesion is designed. Addressing only one function by a module, the number of test cases is

reduced and errors can be more easily predicted and uncovered.

12.5.7 Integrating Testing

Sometimes errors may occur while all modules are putting together and interfacing all. Data may be lost during interfacing; any one module can have an inadvertent, adverse effect on another sub functions, may not produce the desired result, individually acceptable imprecision may be magnified to unacceptable levels, and global data structures can present problems etc. Integration testing is a systematic technique for assembling software while at the same time conducting test to uncover errors associated to interfacing. There are two integration methods: top-down integration and bottom-up integration.

Top-down Integration: Top down integration is the incremental approach of software structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main driver module. Module subroutines are incorporated into the structure in either a *depth-first* or *breadth-first* manner. The integration process is performed in a series of five steps:

- The main control module is used as a test driver, and subs are substituted for all modules directly subordinate to the main control module.
- Depending on the integration approach selected, subordinate subs are replaced one at a time with actual modules.
- Tests are conducted as each module is integrated.
- On completion of each set of tests, another sub is replaced with the real module.
- Regression testing should be conducted to assure that new errors have not been introduced.

  This strategy verifies major decision points early in the test process. In a well-factored software structure, decision making occurs at upper levels in the hierarchy, therefore encountered first. In practice, top-down strategy is relatively uncomplicated but logistical problem can arise, such as when processing at low levels in the hierarchy it is required to adequately test upper levels.

The major *disadvantage* of this approach is the need for subroutines and attendant testing. But the advantage is that the testing of major control functions early.

Bottom-up Integration: It begins assembly and testing with atomic modules. Thus, the need for subroutine testing is eliminated. The implementation steps are:

- Low-level modules are combined into clusters that perform a specific software sub-function
- A driver is added to coordinate test case input and output.
- Each cluster is tested.
- Drivers are removed and clusters are combined, moving upward in the software structure.

  Modules are combined to form cluster and each cluster is tested by using driver. As integration moves upwards, the need for separate test drivers will be very less. Here the program as an entity does not exist until the last module is added. This is the main *disadvantage* of this testing method.

The overall plan for software integration and a description of specific tests are documented in the **test specification**. This specification is a deliverable in the software engineering process and becomes a part of the software configuration.

CHECK YOUR PROGRESS

Q3. Fill in the blanks:

a) In _____ testing the structure of the product is not considered.
b) There are no precise criteria for selecting _____ in black box testing.
c) Black-box testing is concerned with functionality rather than _____ of the program.
d) White box testing is also called _____.
e) _____ testing depends on the capacity of disk or files.

Q4. Write True or False:

a) Storage testing depends on time factor.
b) Black box testing solely depends on requirement specification of the program or module.
c) Performance test refers to the response time of the system being installed.
d) Black-box testing is concerned with implementation of the program.
e) Criterion for the white box test is formal and precise.
f) Unit testing focuses on each individual module.
g) In unit testing two modules cannot be tested in parallel.
h) Integration testing is a systematic technique for assembling software.
i) In top-down integration, integration begins with the main driver module
j) For bottom-up integration need for subroutines are must.
k) In bottom-up approach, low-level modules are combined into clusters and each cluster tested first
l) Test specification is the overall description of integration testing.

m) Before integration testing validation testing is important.

n) Software validation is achieved through a series of white-box tests.

o) Acceptance test is conducted by the end user.

12.6 LET US SUM UP

- Testing is a process of executing a program with the intent of finding an error.
- Testing is generally done in two levels - testing of individual modules and testing of entire system called system testing.
- System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved. Inadequate testing and non-testing system leads to errors.
- Code testing and specification testing are two general strategies of testing software. Code testing examines the logic of the program and specification testing examines program specifications.
- Two different test data are used for testing: Live test data and Artificial test data. Live test data are taken from user's actual files and artificial data are created solely for test purposes.
- Testing is carried out at different levels and at various intervals. Testing each individual module is Unit testing, and the whole system is System testing.
- While integrating different modules, integration testing has to perform. It has two approaches: Top-down integration and Bottom-up integration.
- The overall plan for software integration and a description of specific tests are documented in the **test specification**.
- Two basic testing methods: Black box testing and white box testing are important in any engineering product to test. *Black box testing*, if the product is know, test can be conducted to demonstrate each function to be fully operational; *white box testing*, if the internal workings of a product are known, tests can be conducted to assure that internal operation performs according to specification.
- Apart from the above two testing methods, there are more testing procedures like unit testing, integration testing, stress testing, performance testing etc.

12.7 FURTHER READINGS

- Jalote, P. (2012). *An integrated approach to software engineering*. Springer Science & Business Media.
- Awad, E. M. (1985). *Systems analysis and design*. RD Irwin.
- Mall, R. (2014). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd.

✓ 12. 8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: a) testing

b) time consuming

c) test case

d) code

e) live

Answer to Q2: a) T

b) F

c) F

d) T

e) F

Answer to Q3: a) black box

b) test cases

c) implementation

d) structural testing

e) storage

Answer to Q4: a) F b)T c)T d)F e)T

f) T g)F h)T i)T j)F

k)T l)T m)F n)F o)T

12.9 MODEL QUESTIONS

Q1. There are some special category tests which are not focussed on the normal running of the system. List them.

Q2. Is testing done in every step of developing a system? If yes, why?

Q3. What are the strategies of testing? Explain.

Q4. Distinguish between Unit testing and System testing.

Q5. Why is integration testing required? Discuss the methods of this testing.

Q6. Differentiate between Top-down integration and buttom-up integration method.

Q7. What is structural testing? Distinguish between white box testing and black box testing.

Q8. What is test specification? Write the advantages and disadvantage of bottom-up integration method.

Q9. What are the modules handle at the time of unit testing?

****

UNIT 13: SOFTWARE RELIABILITY AND QUALITY MANAGEMENT

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 Software Reliability

13.3.1 Applications of Software Reliability

13.4 Hardware versus Software Reliability

13.5 Reliability Metrics

13.6 Reliability Growth Modelling

13.7 Statistical Testing

13.8 Software Quality

13.8.1 Software Quality Management System

13.8.2 Evolution of Quality Systems

13.8.3 Product Metrics versus Process Metrics

13.9 Let Us Sum Up

13.10 Answers to Check Your Progress

13.11 Further Reading

13.12 Model Questions

13.1 LEARNING OBJECTIVES

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

- understand basic concepts in software reliability
- differentiate between hardware and software reliability
- describe concepts like reliability metrics and reliability growth modelling
- explain concepts like statistical testing and software quality management

13.2 INTRODUCTION

In the previous unit, we have learned about the different types of software testing. In this unit we will introduce the concept of software reliability and quality management, emphasizing the importance of measuring software reliability through various metrics and the challenges associated with it. It contrasts hardware and software reliability,

outlines reliability metrics and introduces reliability growth modelling. Additionally, the unit covers concepts like software quality factors, quality management systems, product metrics and process metrics. In the next unit, we will explore the concept of software maintenance.

13.3 SOFTWARE RELIABILITY

Reliability of a software product essentially denotes its trustworthiness or dependability. Alternatively, software reliability is defined as the probability of failure-free operation of a software system for a specified time in a specified environment. The key elements of the software reliability include probability of failure-free operation, length of time of failure-free operation and the given execution environment. Failure intensity is a measure of the reliability of a software system operating in a given environment. Example: An air traffic control system fails once in two years.

It is obvious that a software product having a large number of defects is unreliable. It is also clear that the reliability of a system improves, if the number of defects in it is reduced. However, there is no simple relationship between the observed system reliability and the number of latent defects in the system. For example, removing errors from parts of a software which are rarely executed makes little difference to the perceived reliability of the product. It has been experimentally observed by analyzing the behavior of a large number of programs that 90% of the execution time of a typical program is spent in executing only 10% of the instructions in the program. These most used 10% instructions are often called the core of the program. The rest 90% of the program statements are called non-core and are executed only for 10% of the total execution time. It therefore may not be very surprising to note that removing 60% product defects from the least used parts of a system would typically lead to only 3% improvement to the product reliability. It is clear that the quantity by which the overall reliability of a program improves due to the correction of a single error depends on how frequently is the corresponding instruction executed.

Thus, reliability of a product depends not only on the number of latent errors but also on the exact location of the errors. Apart from this, reliability also depends upon how the product is used, i.e. on its execution profile. If it is selected input data to the system such that only the “correctly” implemented functions are executed, none of the errors will be exposed and the perceived reliability of the product will be high. On the other hand, if the input data is selected such that only those functions which contain errors are invoked, the perceived reliability of the system will be very

low.

The reasons why software reliability is difficult to measure can be summarized as follows:

- The reliability improvement due to fixing a single bug depends on where the bug is located in the code.
- The perceived reliability of a software product is highly observer-dependent.
- The reliability of a product keeps changing as errors are detected and fixed.

13.3.1 Applications of Software Reliability

Let us look at the applications of software reliability.

- **Evaluation and comparison of software engineering technologies:** Software reliability helps assess the effectiveness of different software engineering technologies and practices. It enables organizations to determine:
    - The cost involved in adopting a particular technology.
    - The benefits gained from the technology in terms of improved software quality and reduced development or maintenance costs.
    - The overall return on investment (ROI) achieved through the adoption of the technology.
- **Measuring the progress of system testing:** Failure intensity serves as an indicator of the current quality of the software system. A high failure intensity suggests that more testing and debugging activities are required before the software can be considered reliable.
- **Controlling the system in operation:** Software reliability helps monitor and control the performance of a system after deployment. Modifications made for maintenance, enhancements, or bug fixes can influence the reliability of the software, making continuous reliability assessment essential.
- **Gaining better insight into software development processes:** Quantitative reliability and quality measurements provide valuable information about the effectiveness of software development practices. These measurements help identify strengths and weaknesses in the development process, enabling organizations to improve quality assurance and process management.

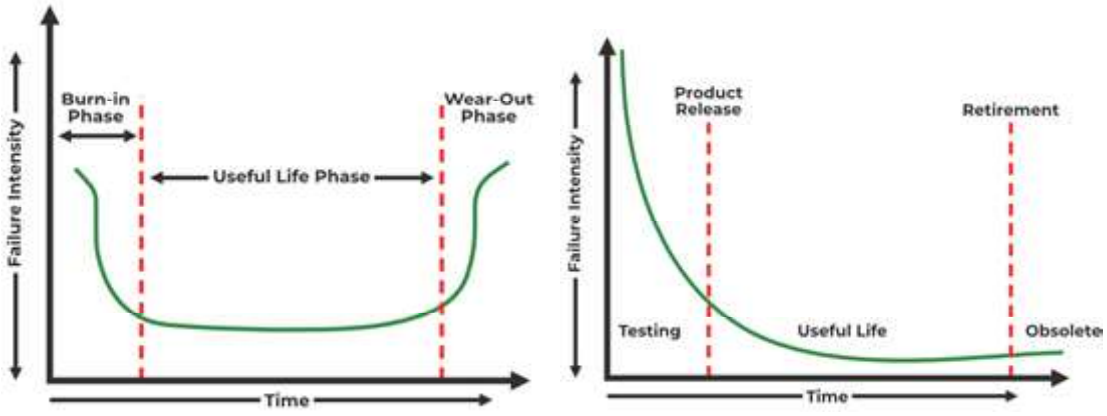
13.4 HARDWARE VERSUS SOFTWARE RELIABILITY

Hardware reliability is the probability that the ability of the hardware to perform its function for some period of time. It may change during certain periods such as initial burn-in or the end of useful life. Some features of hardware reliability are listed below:

- It is expressed as Mean Time Between Failures (MBTF).
- Hardware faults are mostly physical faults.
- Thorough testing of all components cuts down on the number of faults.
- Hardware failures are mostly due to wear and tear.
- It follows the Bathtub curve principle for testing failure.

Software reliability is the probability that the software will operate failure-free for a specific period of time in a specific environment. It is measured per some unit of time. Some features of software reliability are listed below:

- Software Reliability starts with many faults in the system when first created.
- After testing and debugging enter a useful life cycle.
- Useful life includes upgrades made to the system which bring about new faults.
- The system needs to then be tested to reduce faults.
- Software reliability cannot be predicted from any physical basis, since it depends completely on the human factors in design.



(a) Hardware Product **(a) Software Product**

Fig.13.1: Change in failure rate of a product

The change of failure rate over the product lifetime for a typical hardware and a software product are sketched in Fig. 13.1. For hardware products, it can be observed that failure rate is high initially but decreases as the faulty components

are identified and removed. The system then enters its useful life. After some time (called product life time) the components wear out, and the failure rate increases. This gives the plot of hardware reliability over time its characteristics "bath tub" shape. On the other hand, for software the failure rate is at it's highest during integration and test. As the system is tested, more and more errors are identified and removed resulting in reduced failure rate. This error removal continues at a slower pace during the useful life of the product. As the software becomes obsolete no error corrections occurs and the failure rate remains unchanged.

13.5 RELIABILITY METRICS

Reliability metrics measure how consistently a software system performs its intended functions **without failure** over a specified period and environment. These metrics help teams evaluate **software quality, stability, and dependability.** 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.

There are six reliability metrics which can be used to quantify the reliability of software products.

1. Rate of occurrence of failure (ROCOF): ROCOF measures the frequency of occurrence of failures. ROCOF measure of a software product is measured by dividing the total number of failures observed by the duration of observation. However, many software products do not run continuously, but deliver certain service when a demand is placed on them.

2. Mean Time To Failure (MTTF): MTTF looks at how much time has elapsed between two failure occurrences, and it's averaged over the total number of failures. The metric only looks at the time interval between the failures, not the time it took to fix the error and get the software back up and running. It's an important metric to understand how long software can perform before failing and gives developers the ability to predict failures better and get ahead of issues moving forward.

3. Mean Time To Repair (MTTR): MTTR is used to measure the average time taken to track the cause of the error and repair it. It's a metric used to understand

how long it takes to fix an error after a failure occurs and is helpful to help teams understand their working process for reliability and come up with ways to improve.

4. **Mean Time To Failure (MTBF):** The MTTF and MTTR metrics can be combined to get the MTBF metric:
    a. MTBF = MTTF + MTTR
    b. Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours. In this case the measurements are real time and not the execution time as in MTTF.
5. **Probability of Failure on Demand(POFOD):** POFOD is used to measure the likelihood of the system failing when services are requested. It measures the possibility of the system failing when receiving a service request. It can be used in systems where services are requested at an infrequent pace. It's calculated as failures/requests over a specific time interval.
6. **Availability:** Availability of a system is a measure of how likely shall the system be 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.

Limitations of reliability metrics of software products: All the above reliability metrics suffer from several shortcomings as far as their use in software reliability measurement is concerned. One of the reasons is that these metrics are centered around the probability of occurrence of system failures but take no account of the consequences of failures. That is, these reliability models do not distinguish the relative severity of different failures. Failures which are infrequent and whose consequences are not serious are in practice of little concern in the operational use of a software product. These types of failures can at best be minor irritants. On the other hand, more severe types of failures may render the system totally unusable. In order to estimate the reliability of a software product more accurately, it is necessary to classify various types of failures.

A scheme of classification of failures are as follows:-

- **Transient Failures:** These failures occur only for specific input values or under specific, rare circumstances. If the same function is called with different inputs, the failure may not occur.
- **Permanent Failures:** These failures occur for all input values. The failure

persists until the underlying fault is fixed, such as correcting code, replacing hardware, or repairing the system.

- **Recoverable Failures:** When this failure occurs, the system can automatically recover to a functional state without needing to be shut down or restarted, often without human intervention.
- **Unrecoverable Failures:** These are severe failures that bring down the system. The system cannot recover on its own and usually requires a full restart or human operator intervention.
- **Cosmetic Failures:** These failures are minor issues that cause irritation or inconvenience but do not cause incorrect results or disrupt the overall functionality. A common example is needing to click a button twice to trigger a Graphical User Interface (GUI) action.

13.6 RELIABILITY GROWTH MODELLING

A reliability growth model is a mathematical model of how software reliability improves as errors are detected and repaired. A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained. Thus, reliability growth modelling can be used to determine when to stop testing to attain a given reliability level. Although several different reliability growth models have been proposed, in this unit we will discuss only two simple reliability growth models.

1. Jelinski and Moranda Model

This simplest reliability growth model is a step function model where it is assumed that the reliability increases by a constant increment each time an error is detected and repaired. Such a model is shown in Fig. 13.2. However, this simple model of reliability which implicitly assumes that all errors contribute equally to reliability growth, is highly unrealistic since it is already known that correction of different types of errors contribute differently to reliability growth.

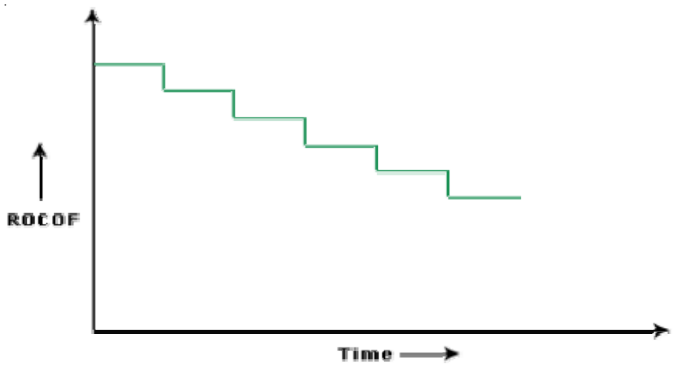


Fig. 13.2: Step function model of reliability growth

2. Littlewood and Verall's Model

This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors. It also models the fact that as errors are repaired, the average improvement in reliability per repair decreases (Figure. 13.3). It treat's an error's contribution to reliability improvement to be an independent random variable having Gamma distribution. This distribution models the fact that error corrections with large contributions to reliability growth are removed first. This represents diminishing return as test continues.

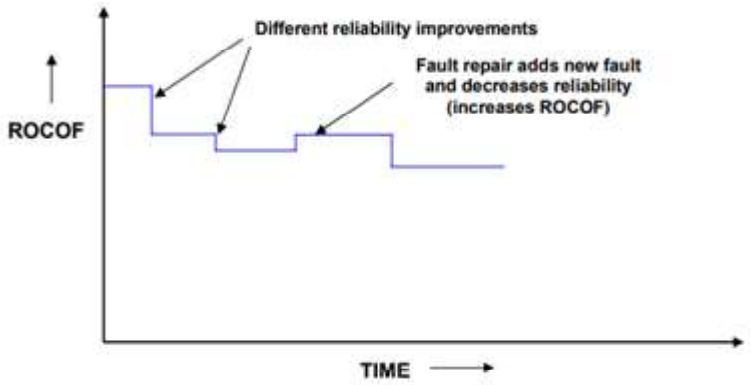


Fig. 13.3: Random-step function model of reliability growth

13.7 STATISTICAL TESTING

Statistical testing is a testing process whose objective is to determine the reliability of the product rather than discovering errors. The test cases are designed for statistical testing with an entirely different objective from those of conventional testing. To carry out statistical testing, we need to first define the operation profile of the product.

Operation prole: Different categories of users may use a software product for very different purposes. We can define the operation profile of a software as the probability of a user selecting the different functionalities of the software. If we denote the set of various functionalities offered by the software by $\{f_i\}$, the operational profile would associate each function $\{f_i\}$ with the probability with which an average user would select $\{f_i\}$ as his next function to use. Thus, we can think of the operation profile as assigning a probability value p_i to each functionality f_i of the software.

Steps in Statistical Testing:

The first step is to determine the operation profile of the software. The next step is to generate a set of test data corresponding to the determined operation profile. The third step is to apply the test cases to the software and record the time between each failure. After a statistically significant number of failures have been observed, there liability can be computed.

For accurate results, statistical testing requires some fundamental assumptions to be satisfied. It requires a statistically significant number of test cases to be used. It further requires that a small percentage of test inputs that are likely to cause system failure to be included. Now let us look at the implications of these assumptions. It is straight forward to generate test cases for the common types of inputs, since one can easily write a test case generator program which can automatically generate these test cases. However, it is also required that a statistically significant percentage of the unlikely inputs should also be included in the test suite. Creating these unlikely inputs using a test case generator is very difficult.

13.8 SOFTWARE QUALITY

Traditionally, the quality of a product is defined in terms of its fitness of purpose. A good quality product does exactly what the users want it to do, since for almost every product, fitness of purpose is interpreted in terms of satisfaction of the requirements laid down in the SRS document "Fitness of purpose" is not a wholly satisfactory definition of quality for software products.

Even though it may be functionally correct, we cannot consider it to be a quality product, if it has an almost unusable user interface. The modern view of a quality associates with a software product several quality factors (or attributes) such as the following:

- **Portability**: A software product is said to be portable, if it can be easily made to work in different hardware and operating system environments, and easily interface with external hardware devices and software products.
- **Usability**: A software product has good usability, if different categories of users (i.e., both expert and novice users) can easily invoke the functions of the product.
- **Reusability**: A software product has good reusability, if different modules of the product can easily be reused to develop new products.
- **Correctness**: A software product is correct, if different requirements as specified in the SRS document have been correctly implemented.
- **Maintainability**: A software product is maintainable, if errors can be easily corrected as and when they show up, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.

13.8.1 Software Quality Management System (SQM)

Software Quality Management ensures that the required level of quality is achieved

by submitting improvements to the product development process. It aims to develop a culture within the team and it is seen as everyone's responsibility. Software Quality management should be independent of project management to ensure independence of cost and schedule adherences. It directly affects the process quality and indirectly affects the product quality.

Components of Software Quality Management System:

The quality system activities encompass the following:

- Auditing of projects to check if the processes are being followed.
- Collect process and product metrics and analyse them to check if quality goals are being met.
- Review of the quality system to make it more effective.
- Development of standards, procedures and guidelines.
- Produce reports for the top management summarising the effectiveness of the quality system in the organisation.

A good quality system must be well documented. Without a properly documented quality system, the application of quality controls and procedures become ad hoc, resulting in large variations in the quality of the products delivered. The ISO 9000 standard provides guidance on how to organize a quality system.

13.8.2 Evolution of Quality System

Quality systems of organizations have undergone **four stages of evolution** as shown in Fig. 13.4.

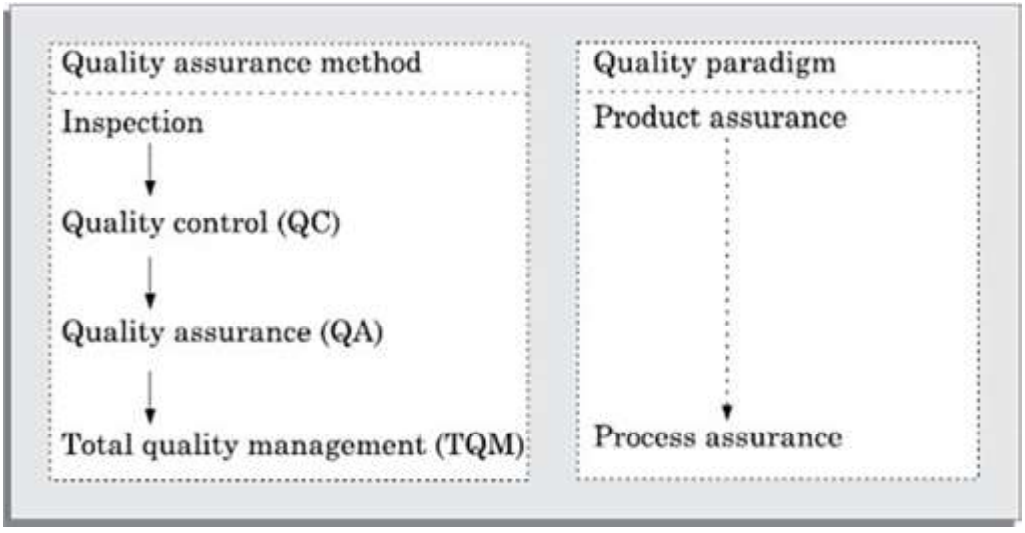


Figure.13.4: Evaluation of quality system and corresponding shift in the quality paradigm

1. Inspection method: The initial product inspection method gave way to quality control (QC) principles.

2. Quality control (QC): Focuses not only on detecting the defective products and eliminating them, but also on determining the causes behind the defects, so that the product rejection rate can be reduced.

3. Modern quality assurance: The basic premise of modern quality assurance is that if an organization's processes are good and are followed rigorously, then the products are bound to be of good quality. It includes guidance for recognizing, defining, analyzing and improving the production process.

4. Total quality management (TQM): TQM advocates that the process followed by an organization must continuously be improved through process measurements. TQM goes beyond documenting processes to optimizing them through redesign. TQM is business process re-engineering and the quality paradigm has shifted from product assurance to process assurance.

13.8.3 Product Metrics Versus Process Metrics

All modern quality systems lay emphasis on collection of certain product and process metrics during product development. The user assesses the quality of a software product based on its external attributes, whereas during development, the developers assess the product's quality based on various internal attributes. We can also say that during development, the developers can ensure the quality of a software product based on measurement of the relevant internal attributes. The internal attributes may measure either some aspects of the product or of the development process (called process metrics).

Let us first understand the basic differences between product and process metrics.

- Product metrics help measure the characteristics of a product being developed. Examples of product metrics are:
  - LOC and function point are used to measure size
  - PM (person- month) metric is used to measure the effort required to develop the software.
  - Time required to develop the product is measure in months.
  - Time complexity of the algorithms, etc.
- Process metrics help measure how a process is performing.
  - Examples of process metrics are review effectiveness, average number of defects found per hour of inspection, average defect correction time,

productivity, average number of failures detected during testing per LOC, number of latent defects per line of code in the developed product.

CHECK YOUR PROGRESS

Choose the correct answer:

Q1. Which of the following is not a phase of “bathtub curve” of hardware reliability?

a) Useful Life

b) Burn-in

c) Wear-out

d) Time

Q2. How is reliability and failure intensity related to each other?

a) direct relation

b) inverse relation

c) no relation

d) none of the mentioned

Q3. How many product quality factors are proposed in McCall quality model?

a) 2

b) 3

c) 11

d) 8

Q4. Which one of the following is not a software quality model?

a) ISO 9000

b) McCall model

c) Boehm model

d) ISO 9126

Q5. What is MTTF ?

a) Maximum time to failure

b) Mean time to failure

c) Minimum time to failure

d) None of the mentioned

Q6. How is software reliability defined?

a) time

b) efficiency

c) quality

d) speed

Q7. How many levels are present in CMM?

a) three

b) four

c) five

d) six

Q8. Which level of CMM is for process management?

a) Initial

b) Repeatable

c) Defined

d) Optimizing

Q9. Failure In Time (FIT) is another way of reporting

a) MTTR

b) MTTF

c) MTSF

d) MTBF

Q10. Mean Time To Repair (MTTR) is the time needed to repair a failed hardware module.

a) True

b) False

13.9 LET US SUM UP

- Software reliability is defined as the probability of failure-free operation of a software system for a specified time in a specified environment.
- Hardware reliability is the probability that the ability of the hardware to perform its function for some period of time.
- Software reliability is the probability that the software will operate failure-free for a specific period of time in a specific environment.
- Reliability metrics measure how consistently a software system performs its intended functions without failure over a specified period and environment.
- A reliability growth model is a mathematical model of how software reliability improves as errors are detected and repaired.

- Statistical testing is a testing process whose objective is to determine the reliability of the product rather than discovering errors.
- Software Quality Management ensures that the required level of quality is achieved by submitting improvements to the product development process.
- Product metrics help measure the characteristics of a product being developed.
- Process metrics help measure how a process is performing.

13.10 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: d

Answer to Q2: b

Answer to Q3: b

Answer to Q4: a

Answer to Q5: b

Answer to Q6: a

Answer to Q7: c

Answer to Q8: 8

Answer to Q9: d

Answer to Q10: a

13.11 FURTHER READING

- Mohapatra, H., & Kumar, A. (2025). *Fundamentals of software engineering*. BPB Publications.
- Cole, E. (2011). *Network security bible*. John Wiley & Sons.
- Mall, R. (2018). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd..
- Fairley, R. E. (2017). Software Engineering Concepts. McGraw Hill Education
- Jalote, P. (2005). *An integrated approach to software engineering*. Boston, MA: Springer US.
- https://www.geeksforgeeks.org/software-engineering/software-engineering-software-quality/
- https://ebooks.inflibnet.ac.in/csp8/chapter/software-reliability/

13.12 MODEL QUESTIONS

Q1. What is software reliability?

Q2. What are the key attributes of reliable software?

Q3. How does reliability relate to software quality?

Q4. What is MTBF (Mean Time Between Failures) and MTTR (Mean Time To Repair)?

Q5. What are the important differences between process metrics and project metrics?

Q6. Differentiate Software and Hardware reliability.

Q7. What is failure rate, and how is it calculated?

Q8. What are reliability growth models?

Q9. What is statistical testing? In what way is it useful during software development?

Q10. Explain in the different steps of statistical testing?

UNIT 14 : SOFTWARE MAINTENANCE AND QUALITY CONTROL

UNIT STRUCTURE

14.1 Learning Objectives
14.2 Introduction
14.3 Software Maintenance
14.4 Types of Software Maintenance
15.5 Software Reverse Engineering
    14.5.1 Characteristic of Reverse Engineering
14.6 Maintenance Cost
14.7 Let Us Sum Up
14.8 Further Readings
14.9 Answers To Check Your Progress
14.10 Model Questions

14.1 LEARNING OBJECTIVES

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

- define software maintenance
- describe the importance of software maintenance
- describe types of software maintenance
- define software reverse engineering
- learn about maintenance cost of software.

14.2 INTRODUCTION

In the previous units, we have dealt with different topics like software testing and software reliablity and quality management. In this unit we will discuss software maintenance and the different types of software maintanence. Topics like software reverse engineering and maintenance of cost are also some of the items being explained here. In the next unit, we will explore the concepts related to CASE tools.

14.3 SOFTWARE MAINTENANCE

Software maintenance is one of the most essential parts of Software Development Life Cycle (SDLC). The maintenance phase of the software life cycle is the time period in

which a software product performs useful work. Typically, the development cycle for a software product spans one to two years, while the maintenance phase spans five to ten years. It is well established that the maintenance activities consume a large portion of the total life cycle budget. According to Lehman's first law,a software product must change continually or become progressively less useful. Therefore,it is needed for all the modification and updation done after the delivery of software products to the user. It is also a broad activity which includes error corrections,enhancements of capabilities, deletion of obsolete capabilities and optimization of software products. Software maintenance process is expensive and risky and also it is very challanging.

Some of the reasons why we need software maintenence are given below:

- Software updations and modifications are required for new policies and due to changes of user requirements and marketing policies.
- Time to time for client requirements, customer may ask new features in the software product.
- Host modification needs software updation and modifications to adjust the changes of required harware or plateform specifications.
- Some times organizational changes also need software updation and modifications to adjust the user requirements with new environments.
- To improve system efficiency and throughput.
- To modify the components of software product.
- To optimize the code to run faster.

14.4 TYPES OF SOFTWARE MAINTENANCE

In a software lifetime,types of maintenance may be based on its nature. Software maintenance can be classified into four categories based on its characteristics as given below:

- ***Corrective -*** It is the reactive and essential modification to correct problems as well as errors in the software product. It means repairing processing or performance failures because of previously uncorrected problems and errors.
- ***Adaptive -*** It is one type of modification which is utilised to keep it usable in a chnaged or changing environment. It means changing the program function which is done to adapt new external changed environments.
- ***Perfective -*** It improves performance and maintenability. It means enhancing the performance or modifying the problems to respond to the user's additional needs.
- ***Preventive -*** It is one type of modification to detect and correct the latent faults.Preventive maintenance is the process by which we prevent our system from being obsolete.It involves the concept of re-engineering and reverse

engineering in which an old system with old technology is re-engineered by using new technology. It also includes modification and updations to prevent future problems occuring to the software product.

14.5 SOFTWARE REVERSE ENGINEERING

Reverse engineering is a process of design recovery. Reverse engineering tools extract data and architectural and procedural design information from an existing program. It becomes important, since lots of software products lack in proper documentation and are highly unstructured. Maintenence activities cannot be performed without a complete understanding of the software system. The main aim of reverse engineering is to recover information from the existing code. It is a process of analysing software with a view to understanding its design and specification. In reverse engineering, source code and executable code are the input.It may be part of a re-engineering process but may also be used to respecify.

The reverse engineering is also the process of generating representations that are implementation independent, starting from code. It is opposite of normal forward engineering process. The main objectives of the reverse engineering process are as follows:

(i) It helps the companies to understand the complexities of the system
(ii) It helps the analyst to generate useful lost information about legacy systems
(iii) It can be used to identify reusable components for analysis and future use
(iv) It helps in generating graphical representation of the system from different perspectives e.g. ER diagram, DFD etc.
(v) Reverse Engineering can be used as a part of reengineering process.
(vi) Over a period of time modifications made to the software also result into unexpected problems. The anomalies can be detected using reverse engineering techniques.

14.5.1 Characteristics of Reverse Engineering

There are various characteristics of reverse software engineering are as follows:

- **Abstract Level :** The abstraction level of reverse engineering process and tools used to the sophistication of design information that can be extraxted from source code.
- **Completeness :** The completeness of a reverse engineering process refers to the level of detail that is provided at an abstraction level.
- **Interactivity:** Interectivity refers to the degree to which the human is interacted with automated tools to create an effective reverse engineering.

- **Directionality:** It is both one-way and two-way processes of the reverse engineering. In case of one -way all information extracted from the source code is provided to the software engineer who can use it during any maintenance activity.
- **Extract Abstraction:** The core of reverse engineering process is an activity called extract abstraction. Here the engineer evaluates the old program from the source code and develop a meaningful specification of the processing which is performed, the interface that is applied.

14.6 MAINTENANCE COST

The cost of software maintenance is very high compared to the other phases of Software Developmenmt Life Cycle (SDLC) process. A study on cost estimation of software maintenance has found that the cost of maintenance is as high as 67% of the cost of the entire cost of software development process cycle. On an average, the cost of software maintenance is more that 50% of all SDLC phases.

Some important **software-end** factors aeffecting the maintenance cost are given below:

(i) Structure of software programs.

(ii) Programming language

(iii) Staff availability and reliability.

(iv)Dependent on the external environments

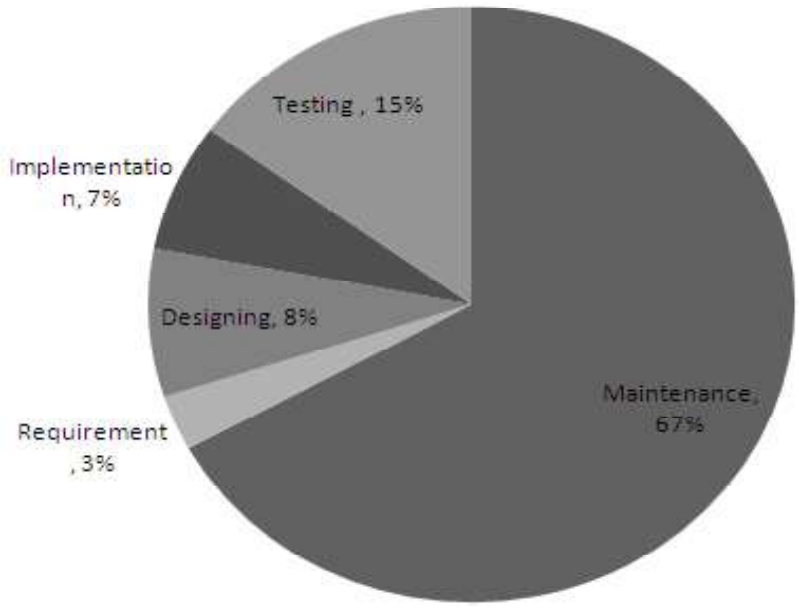


Figure14.1: Result of cost of software maintenance in SDLC

Some other **real-world** factors affecting the maintenance cost are given below:

(i) As technology advances, it becomes costly to maintain the old software products.

(ii) The standard age of any software product is considered upto 10 to 15 years.

(iii) Older software that runs on slow machine with less memory cannot keep themselves challenging against newly coming enhanced software with modern hardware.

(iv) Most of the engineers newly coming use trial and error method to rectify the problems.

Software Maintenance Activities:

It consists of eight (8) steps namely identification and tracing, analysis, design, implementation, system testing, acceptance testing, delivery and finally maintenance management.

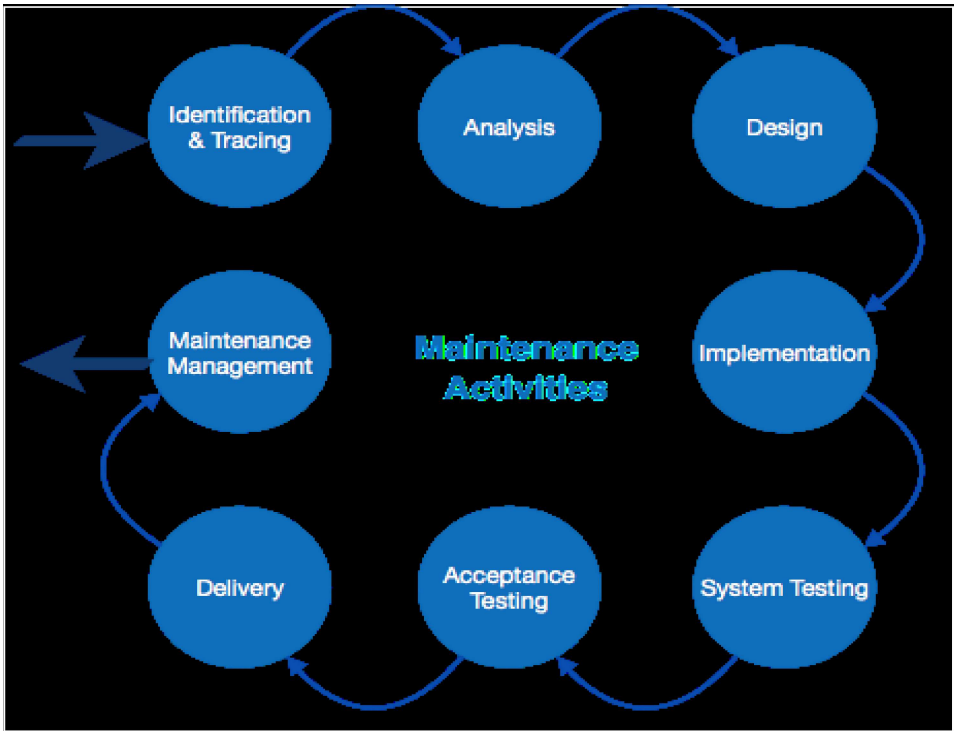


Figure 14.2: Software Maintenance activities

CHECK YOUR PROGRESS

Q1. Fill in the blanks :

a) Software maintenance process is _____ and _____and also it is very _____.

b) Reverse Engineering can be used as a part of_____process.

c) On an average the cost of software maintenance is more that _____ of all SDLC phases.

d) The core of reverse engineering process is an activity called _____.

e) _____ improves performance and maintenability of software.

14.7 LET US SUM UP

- Software maintenance is one of the most essential parts of Software Developmenmt Life Cycle (SDLC). According to Lehman's first law,a software product must chnage continually or else it becomes progressively less useful. Therefore, it needs all the modification and updation to done after the delivery of software products.
- Software maintenance can be classified into four categories based on its charactersitcs are (i) Corrective, (ii) Adaptive, (iii) Perfective and (iv) Preventive.
- Reverse Engineering is the process followed in order to know difficult unknown and hidden information about software system.
- The main aim of reverse engineering is to recover information from the existing code.
- Reverse engineering is also the process of generating representations that are implementation independent, starting from code.
- Reverse engineering is opposite of the normal forward engineering process
- The various characteristics of reverse software engineering are (i) Abstract level,completeness, interactivity,Directionality and extract abstraction.
- A study on cost estimation of software maintenance found that the cost of maintenance is as high as 67% of the cost of the entire cost of software development process cycle.

14.8 FURTHER READINGS

- Waman, S. J. (2004). Software Engineering–Principles and Practice. Tata McGrawHill.
- Leach, R. J. (2016). Introduction to software engineering. CRC Press.
- Jalote, P. (2012). An integrated approach to software engineering. Springer Science & Business Media.

✓ 14.9 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: a) expensive, risky, challanging;

b) reengineering

c) 50%;

d) extract abstraction

e) Perfective

? 14.10 MODEL QUESTIONS

Q1. Define software maintenance.

Q2.What is the need of software maintenance?

Q3. What are the categories of software maintenance? Explain.

Q4. Define software reverse engineering.

Q5. What are the characteristics of reverse software engineering? Explain

Q6. Describe the process of maintenance of cost.

Q7. What are the objectives of software reverse engineering?

******

UNIT 15:CASE TOOLS

UNIT STRUCTURE

15.1 Learning Objectives

16.2 Introduction

15.3 CASE Tools

15.3.1 Need of CASE Tools

15.4 Classification of CASE Tools

15.5 Types of CASE Tools

15.6 Advantages of CASE

15.7 Integrated CASE Environment

15.8 Let Us Sum Up

15.9 Further Readings

15.10 Answers To Check Your Progress

15.11 Model Questions

15.1 LEARNING OBJECTIVES

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

- understand the meaning of CASE Tools
- explain the importance of CASE Tools
- list the different categories of CASE Tools
- describe the intergated CASE environment

15.2 INTRODUCTION

In the previous units, we have discussed in detail about software maintenance. Topics like software reverse engineering and maintenance of cost have also been discussed in detail in the last unit.

In this last unit of the course, we will discuss CASE tools as well as the need for CASE tools and their advantages. The four classifications of CASE tools are described in detail in this unit in addition to the different types of CASE tools along with a description of the integrated CASE environment.

15.3 CASE TOOLS

CASE stands for Computer Aided Software Engineering.It means development and maintenance of software projects or products with the help of various automated software tools.

A **CASE tool** is a generic term used to denote any form of automated support for software engineering. In a more restrictive sense, a CASE tool means any tool which is used to automate some activity associated with software development project. Also, we can define CASE tools as a set of software application programs, which are used to automate (Software Development Life Cycle) SDLC activities.

A CASE tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process.

CASE tools are used by software project managers, analysts and engineers to develop software system smoothly and reliably.

There are a number of CASE tools available to simplify various stages of SDLC such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools to name a few. Some of these CASE tools assist in phase related tasks such as specification, structured analysis, design, coding, testing, etc, and others to non-phase activities such as project management and configuration management.

15.3.1 Need of CASE Tools

A CASE environment facilitates the automation of the step-by-step methodologies for software development. In contrast to a CASE environment, a programming environment is an integrated collection of tools to support only the coding phase of software development.

CASE tools also provide automated methods for designing and documenting traditional structured programming techniques. The ultimate goal of CASE is to provide a language for describing the overall system that is sufficient to generate all the necessary programs needed.

Use of CASE tools accelerates the development of project to produce the desired result and helps to uncover flaws before moving ahead with next stage in software development.

The primary utility of using a CASE tool:

- To increase productivity
- To help produce better quality software at a lower cost

• To develop more reliable and fault torant software product.

15.4 CLASSIFICATION OF CASE TOOLS

CASE tools can be classified into four different dimensions, namely

1. Life-cycle support
2. Integration dimension
3. Construction dimension
4. Knowledge-based CASE dimension

Let us explain the meaning of the first two dimensions here.

1. Life-Cycle Based CASE Tools

This dimension classifies CASE Tools on the basis of the activities they support in the information systems life cycle. They can be classified as Upper CASE tools and Lower CASE tools.

- **Upper CASE Tool**

UpperCASE Tool is a CASE software tool which supports the software development activities upstream from implementation. Upper case tool focuses on the analysis phase but sometimes also on the design phase of the software development lifecycle. For example, diagramming tools, report and form generators, and analysis tools are upper case tools.

- **Lower CASE Tool**

LowerCASE Tool is a CASE software tool that directly supports the implementation (programming) and integration tasks. Lower CASE tools support database schema generation, program generation, implementation, testing, and configuration management.

2. Integration dimension

There are three main CASE Integration dimensions which are as follows :

a) CASE Framework

b) ICASE Tools: Tools that integrate both upper and lower CASE. An automated system development environment that provides numerous tools to create diagrams, forms and reports. It also offers analysis, reporting, and code generation facilities and seamlessly shares and integrates data across and between tools.

c) Integrated Project Support Environment(IPSE)

15.5 TYPES OF CASE TOOLS

There are different types of CASE tools that are available now.The general types of CASE tools are listed below:

1. **Diagramming tools:** The tools that enable system process, data and control structures to be represented graphically.

2. **Computer display and report generators:** It helps prototype how systems look and feel. It makes it easier for the systems analyst to identify data requirements and relationship.

3. **Analysis tools:** It automatically checks for importance, inconsistent, or incorrect specifications in diagrams, forms, and reports.

4. **Central repository:** It enables the integrated storage of specifications, diagrams, reports and project management information.

5. **Documentation Generators:** It produces technical and user documentation in standard formats.

6. **Code generators:** It enables the automatic generation of program and data base definition code directly from the design documents, diagrams, forms, and reports.

15.6 ADVANTAGES OF CASE TOOLS

There are several benefits and advantages to the use of a CASE environment or even isolated CASE tools.

Some of those benefits are given below:

- A key benefit arising out of the use of a CASE environment is cost saving through all development phases.
- Use of CASE tools leads to considerable improvements in quality. This is mainly due to the fact that one can effortlessly iterate through the different phases of software development and the chances of human error are considerably reduced.
- CASE tools help produce high quality and consistent documents.
- CASE tools take out most of the drudgery in a software engineer's work.
- CASE tools have led to revolutionary cost saving in software maintenance efforts.
- Introduction of a CASE environment has an impact on the style of working of a company, and makes it oriented towards the structured and orderly approach.

15.7 INTEGRATED CASE ENVIRONMENT

An environment is a collection of CASE tools and workbenches which supports the software process. CASE environments are classified on the basis of the focus/basis

of integration. These are respectively:

1. Toolkits
2. Language-centered
3. Integrated
4. Fourth generation
5. Process-centered

1. Toolkits:

Toolkits are loosely integrated collections of products easily extended by aggregating different tools and workbenches.

2. Language-centered:

The environment itself is written in the programming language for which it was developed, thus enabling users to reuse, customize and extend the environment.

3. Integrated:

These environments achieve presentation integration by providing uniform, consistent, and coherent tool and workbench interfaces. Data integration is achieved through the repository concept: they have a specialized database managing all information produced and accessed in the environment. Examples of integrated environment are the ICL CADESsystem, IBM AD/Cycle and DEC Cohesion.

4. Fourth-generation:

Fourth-generation environments were the first integrated environments. They are sets of tools and workbenches supporting the development of a specific class of program: electronic data processing and business-oriented applications.

5. Process-centered:

Environments in this category focus on process integration with other integration dimensions as starting points. A process-centered environment operates by interpreting a process model created by specialized tools.

CHECK YOUR PROGRESS

Q1. Fill up the blanks:

a) CASE stands for _____.

b) A CASE environment facilitates the automation of the step-by-step -_____ for software development.

c) CASE tolls helps to produce better quality software at _____.

d)_____ automatically checks for importance, inconsistent, or incorrect specifications in diagrams, forms, and reports.

e) _____ enables the automatic generation of program and data base definition code directly from the design documents, diagrams, forms, and reports.

15.8 LET US SUM UP

- A CASE tool is a generic term used to denote any form of automated support for software engineering. Also we can define CASE tools are set of software application programs, which are used to automate (Software Development Life Cycle) SDLC activities.
- There are number of CASE tools available to simplify the various stages of SDLC such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools to name a few.
- CASE tools can be classified into four different dimensions namely:
  - Life-cycle support
  - Integration dimension
  - Construction dimension
  - Knowledge-based CASE dimension.
- UpperCASE Tool is a CASE software tool which supports the software development activities upstream from implementation.
- LowerCASE Tool is a CASE software tool that directly supports the implementation (programming) and integration tasks.
- There are different types of CASE tools now available:
  - **Diagramming tools:** The tools that enabled system process, data and control structures to be represented graphically.
  - **Computer display and report generators:** It helps prototype how systems look and feel. It makes it easier for the systems analyst to identify data requirements and relationship.
  - **Analysis tools:** It automatically checks for importance, inconsistent, or incorrect specifications in diagrams, forms, and reports.
  - **Central repository:** It enables the integrated storage of specifications, diagrams, reports and project management information.
  - **Documentation Generators:** It produces technical and user documentation in standard formats.
  - **Code generators:** It enables the automatic generation of program and data

base definition code directly from the design documents, diagrams, forms, and reports.

15.9 FURTHER READINGS

- Jalote, P. (2012). *An integrated approach to software engineering.* Springer Science & Business Media.
- Waman, S. J. (2004). *Software Engineering–Principles and Practice.* Tata McGrawHill.
- Leach, R. J. (2016). *Introduction to software engineering.* CRC Press.

15.10 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1:

(i) Computer Aided Software Engineering

(ii) methodologies

(iii) lower cost

(iv) Analysis tools

(v) Code generators

15.11 MODEL QUESTIONS

Q1. What are CASE Tools. Define briefly.

Q2. What is the need of CASE Tools?

Q3. What are the different types of CASE Tools? Describe the briefly.

Q4. What are the advantages of CASE Tools?

Q5. Explain the classification of CASE Tools?

*****