Syllabus

Course : USCS404	TOPICS (Credits : 02, Lectures/Week : 03) Software Engineering

Unit	Details	Lectures
I	**Introduction :** The Nature of Software, Software Engineering, The Software Process, Generic Process Model, The Waterfall Model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Component-Based Development, The Unified Process Phases, Agile Development- Agility, Agile Process, Extreme Programming. **Requirement Analysis and System Modeling :** Requirements Engineering, Eliciting Requirements, SRS Validation, Components of SRS, Characteristics of SRS , Object-oriented design using the UML - Class diagram, Object diagram, Use case diagram, Sequence diagram, Collaboration diagram, State chart diagram, Activity diagram, Component diagram, Deployment diagram. **(Refer Chapters 1, 2, 3 and 4)**	15
II	**System Design :** System/Software Design, Architectural Design, Low-Level Design, Coupling and Cohesion, Functional-Oriented Versus The Object-Oriented Approach, Design Specifications, Verification for Design, Monitoring and Control for Design. **Software Measurement and Metrics :** Product Metrics - Measures, Metrics, and Indicators, Function-Based Metrics, Metrics for Object-Oriented Design, Operation-Oriented Metrics, User Interface Design Metrics, Metrics for Source Code, Halstead Metrics Applied to Testing, Metrics for Maintenance, Cyclomatic Complexity, Software Measurement - Size-Oriented, Function-Oriented Metrics, Metrics for Software Quality. **Software Project Management : Estimation** in Project Planning Process - Software Scope And Feasibility, Resource Estimation, Empirical Estimation Models - COCOMO II, Estimation for Agile Development, The Make/Buy Decision, **Project Scheduling** - Basic Principles, Relationship Between People and Effort, Effort Distribution, Time-Line Charts. **(Refer Chapters 5, 6, 7 and 8)**	15
III	**Risk Management** - Software Risks, Risk Identification, Risk Projection and Risk Refinement, RMMM Plan. **Software Quality Assurance :** Elements of SQA, SQA Tasks, Goals, and Metrics, Formal Approaches to SQA, Six Sigma, Software Reliability, The ISO 9000 Quality Standards, Capability Maturity Model. **Software Testing :** Verification and Validation, Introduction to Testing, Testing Principles, Testing Objectives, Test Oracles, Levels of Testing, White-Box Testing/Structural Testing, Functional/Black-Box Testing, Test Plan, Test-Case Design. **(Refer Chapters 9,10 and 11)**	15

❑❑❑

UNIT I

UNIT II

□□□

UNIT I

CHAPTER 1

Introduction to Software Engineering

Syllabus :
The Nature of Software, Software Engineering.

1.1 Introduction

– Computers are becoming an unavoidable necessity or say the part and parcel of our life. We do most of our day's work using computers and digital devices – to book railway tickets or movie tickets, search for some books on amazon, bank transactions, and any task you name it is done using computers. Therefore it is very important to build these computerized systems in an effective way.

– Building such systems requires certain technical as well as communication skills and capabilities to understand and follow a planned and systematic procedure.

1.2 Software

Q. Define software.

– Software is a set of instructions to acquire inputs and to process them to produce the desired output in terms of functions and performance as determined by the user of the software. It is developed to handle an *Input-Process-Output* system to achieve predetermined goals.

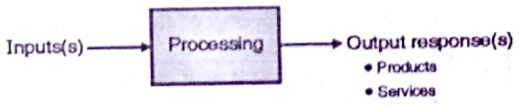


Fig. 1.2.1 : Basic diagram of a software

– **Software encompasses of**
1. Instructions (Computer programs) 2. Documents (Describes programs)
3. Architecture including Data Structures (Enables programs)

– Software includes things such as websites, programs or video games that are coded by programming languages like C or C++.

☞ **Software - a Product and a Vehicle :**

"Software plays a dual role - both as a *product* and a *vehicle* that delivers a product".

☞ **Software is a product :**

1. Delivers computing potential

2. Produces, manages, acquires, modifies, displays, or transmits information. (e.g., software is in a cellular phone or in any computer)

☞ **Software is a vehicle for delivering a product :**

1. Controls other programs (e.g., an operating system)
2. Effects communications (e.g., networking software)
3. Helps build other software (e.g., software tools)

1.2.1 Characteristics of Software

Q. State characteristics of software..

– Software means anything which is not hardware but which is used with hardware, such as windows OS (is system software) in computer (is hardware).

You will get more precise idea about the characteristics of software and how it differs from hardware from the below table.

Table 1.2.1 : Characteristics of software and how it differs from hardware

Sr. No.	Software	Hardware
1.	It is developed or engineered.	It is manufactured.
2.	It doesn't wear out as it is not prone to environmental problems.	It wears out as the time passes due to the affects of dust, vibration, temperature extremes and many such environmental problems.
3.	There are no software spare parts which can be used to replace the software.	When hardware fails, it can be replaced by spare parts.
4.	Software is untouchable. It is the code or instructions that tell a computer/hardware how to operate. It has no substance.	Hardware is a physical device something that you're able to touch and see.
5.	Software is usually generic but it can also be custom built (developed according to the customer specification).	It is manufactured or assembled by using the existing components.
6.	Software is invaluable as it can be installed in any hardware.	Hardware has no value without software in it. If computers (h/w) had no operating system (s/w), then no customer will buy it.
7.	Examples: Microsoft, Windows any operating system that allows you to control your computer. Example 2: Internet browsers, video games, applications like Payroll etc.	Examples: disks, disk drives, display screens, keyboards, printers and chips.

1.2.2 Classes of Software

Software is classified into two types of classes:

- Generic
- Customized

Table 1.2.2 : Difference between generic software and customized software

Sr. No.	Generic Software	Customized Software
1.	Designed for broad customer market.	Designed for specific business purposes.
2.	Is open to market and its specifications are designed by the programmer.	Is an s/w and its specifications are designed according to a particular firm or organization; it is not open for all.
3.	Examples: ERP/CRM, CAD/CAM packages, OS, System Software.	Examples: Legacy systems, Process control systems, Traffic management system and Hospital management system.
4.	Generic software development is done for general purpose audience.	Custom Software Development is done to satisfy a particular need of a particular client.
5.	Requirements and specifications of this software are managed by the developer.	Managed by the customer and influenced by the practices of that industry.
6.	General Purpose software development is tough as compared with Custom made by the design and marketing point of view.	By Buyer's point of view, he prefers to have a Custom Made application developed rather than buying a General Purpose software as he gets the application done that exactly matches his requirements.
7.	In General Purpose application design and development, you need to imagine what an end-user requires. Market Surveys and general customer demand analysis may help in such designs.	In Custom Made application, you have a specific end – user. Understanding his need and analyzing it to get the best out of it helps in such designs.

Syllabus Topic : The Nature of Software

1.3 The Nature of Software

Software is classified into 7 categories as below :

1. System software

It directly interacts with computer hardware. It is generic software.

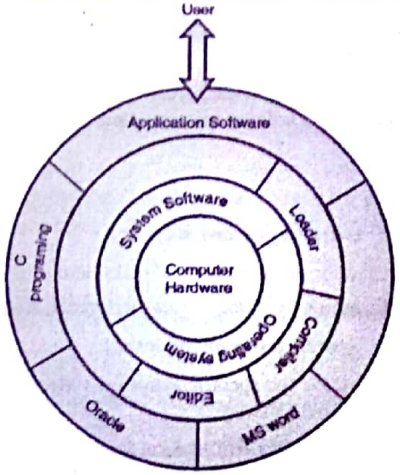


Fig. 1.3.1 : Layers of the computer software

System software is classified into three categories :

(1) Operating system
(2) System support software
(3) System development software.

(1) **Operating system :** Operating system acts as an interface between user and the hardware, and provides different services to users.

Examples : DOS, Windows-XP, LINUX etc.

(2) **System support software :** System support software manages hardware more efficiently.

Examples : Drivers of the IO devices or Antivirus software.

- Drivers of the IO device consist of interfacing programs which can interacts with IO device. Every IO device has its own driver programs.
- Antivirus programs remove viruses from the systems and some time recover systems from major data loss.

(3) **System development software :** System development software supports programming development environment to user.

Examples : Editor, Pre-Processors, Compiler, Interpreter, Loader etc.

- Editor is used to create programs as well as it is used for modification of existing programs.
- Pre-processor works before the use of translators (that means compiler or editors) to replace some segments of code with some other segment. It is also called as an expansion.
- Compilers can translate high level programs into low level programs.
- Interpreters can translate line by line high level programs into low level programs.
- Loaders are the software which can load object codes into the main memory and execute it.

2. Application software

Application software is only designed to solve user problems as per user's requirement.

Application software can generic or customized. Application software is classified into two categories :

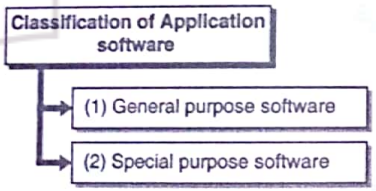


Fig. C1.1 : Classification of Application software

➜ **(1) General purpose software :**

It used for much number of tasks, and provides many features. It is generic software.

Examples : Word Processor, Oracle, Excel etc.

➜ **(2) Special purpose software :**

It designed for specific purposes only. User programs come under special purpose software. It is customized software.

Examples : Pay Roll system for specific company, Tax Calculation Software etc.

Q. Differentiate between system software and application software

Table 1.3.1 : Comparison of system software with application software

Sr. No.	System software	Application software
1.	Primarily concerned with the efficient management of the computer system.	Primarily concerned with the solution of some problems, using the computer as a tool.
2.	System programs are intended to support the operation and use of the computer itself.	The focus is on the application, not on the computing system.
3.	Is usually related to the architecture of the machine on which it is to run. Thus, system programmer must have knowledge of computer architecture.	Is usually related to the detailed knowledge of the Language and environment. Thus, application programmer must have detailed knowledge of high level programming language which is used to develop application program and environment (operating system) on which they are to run.
4.	It is machine dependent.	It is machine independent.
5.	Used to develop new system programs and using Bootstrapping we can make them portable.	Used to develop new application programs and using cross compiler i.e. again system programs, we can make them portable.
6.	Examples of system software are Assembler, linker, loader, compiler etc.	Examples of Application software are Paint, Photoshop, Microsoft access etc.

3. Engineering/Scientific software

- This type of software deals with processing requirements in specific applications. These are specially used for drawing, drafting, modeling, load calculations and analyzing of engineering and statistical data for interpretation and decision making.
- **Examples :** CAD, CAM, CAE software's. Computer-aided design (CAD) software, computer-aided manufacturing (CAM) software and computer-aided engineering (CAE) software is used in mechanical, electrical or electronic design; simulation, drafting, and engineering; and analysis and manufacturing. CAD, CAM, CAE and design software runs on mainframes, general-purpose workstations, and personal computers (PCs).

4. Embedded software

- This type of software is embedded into hardware as a part of larger systems to control its various functions. This type of software is embedded in Read-Only-Memory (ROM) of the systems/products.
- **Examples :** Keypad control software embedded in a microwave oven or washing machine where there is a need to input, identify the status, decide and take action. These are also called as Intelligent Software.

5. Product-line software

- Software product lines refers to software engineering methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production.

- A Product Line Software is a set of software products that share common features but are each different in some way e.g. they may be developed for specific customers or markets (e.g. inventory control products), or, for embedded software (word processors, spreadsheets, computer graphics, personal and business financial applications).

6. Web applications

- A web application is accessed via web browser over a network such as the Internet or an intranet. It is coded in browser-supported languages such as HTML, JavaScript, JSP ASP, PHP. .
- The first-generation web applications allowed the business to post information publicly. Thus, this information is seen to anyone with a Web browser and Internet access. The problem with the first-generation web application is that the information is static.
- The second-generation web applications allowed the users to do interactive queries against existing databases from the Web application. **"Web 2.0"** refers to the second generation of web development and web design. It allows communication, information sharing, interoperability, user-centered design and collaboration on www.
- Third-generation Web application is a powerful business tool for organizations in their Electronic Commerce (EC) efforts.

7. Artificial intelligence software

- This software makes use of non numerical algorithms which use data generated in the system to solve complex problems that are not amenable to problem solving procedures and require specific analysis and interpretation of the problem to solve it.
- **Examples:** robotics, expert systems, artificial neural networks and computer games. All these software's can run either in real-time mode or offline mode. This software's can be shared free or charged by usage. Thus, they are also called Shareware and Freeware. Some of these types of software are run on desktop computers, others on mainframe and mini computers, some are exclusively designed for web, network or the internet.

1.3.1 The Evolving Role of Software

- Software evolution is the term used in Software engineering to refer to the process of developing the software initially and then repeatedly updating it for various reasons.
- As the computer age started in 1960's, software had acquired a great importance in information technology. In the initial stage, the scientists and engineers were the only users of the computers and they only wrote the programs for developing the software. That means, user and the programmer were the same person.
- It's genuine that - as the computing technology, data storage process, communication, networking and programming languages advance, software also undergoes a change.
- The software written in 1960's are totally different from what they are now in all its specifications, such as lines of code, program structure and architecture. Today's software are shorter, smarter and more efficient compared to older versions.
- Nowadays, software includes the source code, executables, design specifications, functions and system's user manuals, installation and implementation manuals to understand the software system.
- Today's trend is to develop the software components that are platform independent.

- The advances in software came hand in hand with more advances in computer hardware. In the mid 1970s, the microcomputer was introduced. This in turn led to the now famous Personal Computer or PC and Microsoft Windows.
- The Software Development Life Cycle (SDLC) also started to appear as a 'centralized construction of software' in the mid 1980s.
- Open-source software started to appear in the early 90s in the form of Linux.
- The Internet and World Wide Web hit in the mid 90s.
- Distributed Systems gained a way to design systems.
- Programmers collaborated and wrote the Agile Manifesto that favored more light weight processes to create cheaper and timelier software.
- Now, the era is of Expert Systems, AI, Neural Networks, Parallel computing and Network computers.

The 'early era -one programmer' is replaced by teams of software engineers, each focusing on different parts of the technology required to build the complete software. And yet the questions asked to today's software programmers are same as that asked to the early era lone programmers:

- Takes more time to develop
- Development cost is high
- Defects found even after delivery
- More that estimated time and budget spent on maintaining the software

These concerned questions with the software development have led to the idea of software engineering practice.

Syllabus Topic : Software Engineering

1.4 Software Engineering

Q. Define Software engineering and its Objectives.

Definition

Software Engineering is a systematic, scientific and disciplined approach towards the development, functioning and maintenance of the software.

As described in the above definition, the systematic and scientific approach in software engineering can be defined as :

- Understanding of customer requirements both by customer and developer.
- Use of structured methodology to gather customer requirements and its analysis to arrive at the Software Requirements Specification(SRS)
- Right estimation of resource, efforts and costs.
- Use of developmental and testing methodology to ensure the quality of software.
- Ease of installation, demonstration and implementation of software by the customer and users.

Such an Engineering approach is required to ensure that the software is designed with the correct choice of technology and architecture to :

- Achieve customer satisfaction
- Ensure on-time delivery

- Be developed within the budget cost
- Provide ease of maintenance to meet changing requirements.
- Else it leads to :
- Unreliable and poor quality software development.
- Late delivery to the customer.
- Difficulty in maintenance
- Lot of expenses in development process ultimately expensive software.

Primary goal of software engineering : to provide quality software at low cost on planned delivery date. Software engineering involves various phases such as project planning, systematic analysis and design, testing and maintenance.

Basic Objectives of Software Engineering are :

- Define software development plan
- Manage software development activities
- Design the proposed product
- Code/develop the product
- Test the product modules
- Integrate the modules and test the system as a whole
- Maintain the product

Software Engineering is the part of a larger subject called *Systems Engineering* which considers issues like hardware platform, operating system, and interoperability between platforms, performance, scalability and upgrades to develop the software. While considering these issues, it uses Computer Aided Software Engineering (CASE) tools (e.g. ArgoUML, Case Studio 2, MagicDraw), software quality testing tools, code generators (e.g. XMLSpy, UModel), and report writers. Thus, a good software engineer must know how to use these tools to develop quality software.

Key challenges of Software Engineering :

- **Heterogeneity :** to use the development techniques that can cope with diverse platforms and execution environments.
- **Delivery :** to use development techniques that lead to faster software delivery.
- **Trust :** to use development techniques that can build trust in users' opinion.

1.5 Software Engineering – A Layered Technology

Q. Explain the layered technology of Software Engineering.

Fig. 1.5.1 : Layers of software engineering

1.5.1 Quality Focus

Software Engineering rests on an organizational commitment to *quality*, which leads to continuous improvements in the Software engineering process. *Focus on quality is always the primary goal of software engineering*. (Quality of Software is explained in section 1.4). In short, we can say that the software is qualitative if it is read and understood easily, if it is modifiable and accommodates new requirements, if it has customer satisfaction and is completed in time within budget.

- The Primary Goal of Any Software Process : High Quality
- Remember : High quality = project timeliness
- Why ? Less rework!

1.5.2 Software Process

Q. What is the need of Software Process ?

When building a product, it's important to go through some predictable steps i.e. called *process* that helps you to create a timely and high quality product. A *process* is the foundation of software engineering. It defines a framework that must be established to ensure effective delivery of Software Engineering technology. The key use of a *process* is :

- Producing models, documents, data, forms and reports.
- Establishing milestones.
- Software Quality Assurance (SQA).
- Software Configuration (change) Management (SCM).

1.5.3 Software Engineering Methods

Method is an ordered way of developing a software. This includes the suggestions for the process to be followed, the notations to be used, the rules governing the system descriptions and the design guidelines

It provides the technical "how to's" for building a software. It includes how to implement the following generic processes :

- Communication
- Requirements Analysis
- Design
- Program Construction
- Testing
- Maintenance

Methods are organized ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.

1.5.4 CASE Tools

- They provide automated or semi-automated support for the process and the methods. For example; CASE tool.
- CASE represents Computer Aided Software Engineering tool used in software development process.
- A CASE tool is a computer-aided product specially designed to support the software engineering activities within a software development process.

Examples of Case tools are :

- Code generation CASE tools - Visual Studio .NET
- Code analysis CASE tools -Borland Audits
- CASE tools used for development of data models -UML editors
- Cleaning up code CASE tools -Refactoring tools
- Bug tracker CASE tool
- Version control CASE tool -CVS, etc.

Benefits

- Improves the productivity
- Generates small parts of code automatically
- Improves software quality
- Can be integrated with other tools say, with code editor to work with coding.

Review Questions

Q. 1 Define software and state its characteristics.

Q. 2 Differentiate between system software and application software.

Q. 3 Define Software engineering and its Objectives.

Q. 4 Explain the layered technology of Software Engineering.

Q. 5 What is the need of Software Process ?

Q. 6 Identify the following into types of software.

Operating System, Compiler, Testing Software, Sales Analysis, 2D Design Software, Statistical Analysis, Spread Sheet Application, Graphical Analysis and Design, Expert Systems, Automatic Operations Software on POW in Oven/Washing Machines/Gas Stations.

□□□

UNIT I

CHAPTER 2

Software Process

Syllabus

The Software Process, Generic Process Model, The Waterfall Model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Component-Based Development, The Unified Process Phases, Agile Development- Agility, Agile Process, Extreme Programming.

Syllabus Topic : The Software Process

2.1 Software Process

Q. Explain in brief the various types of Software Process.

A software process identifies a set of activities that are applicable to the development of any software project, regardless of their size or complexity.

A software process is a collection of work activities, actions and tasks that are to be performed when some software project is to be developed.

Software process can be categorized into :

1. Generic Process model – represents a framework activity populated by a set of software engineering activities. It includes :
   - Framework activities
   - Umbrella activities
2. Personal and Team Process models – This model helps in creating a software that best fits either the personal needs of the user or that meets the broader needs of a team. It includes :
   - Personal Software Process (PSP)
   - Team Software Process (TSP)
3. Prescriptive Process models – provides an ordered structure and an effective roadmap to software engineering work. It includes :
   - Waterfall model – Incremental model – RAD model
   - Evolutionary Process models – Prototyping – Spiral model
   - Concurrent Development model

Prescriptive models define a discrete set of activities and actions to accomplish all tasks of the software with milestones, which is used to develop the software. These Process models may not be perfect but they give very good guidance in software development process.

This model is important because,

- It is also referred as rigorous model and provides stability.
- It describes a unique set of frame work activities and organizes them into a process flow.
- These actions are used to create work product to accomplish to meet development goal.
- It finds out- the nature of project, whether is suitable for the people using it, whether it is suitable for the environment, where is it implemented.
- It prescribes a set of process elements, framework activities, software engineering actions tasks, work products quality assurance, change control mechanism of each project.

Syllabus Topic : Generic Process Model

2.2 Generic Process Model

Q. List the umbrella activities followed in generic process model.

This model defines a set of *umbrella activities* which are also a must for any software engineering process as shown in the Fig. 2.2.1.

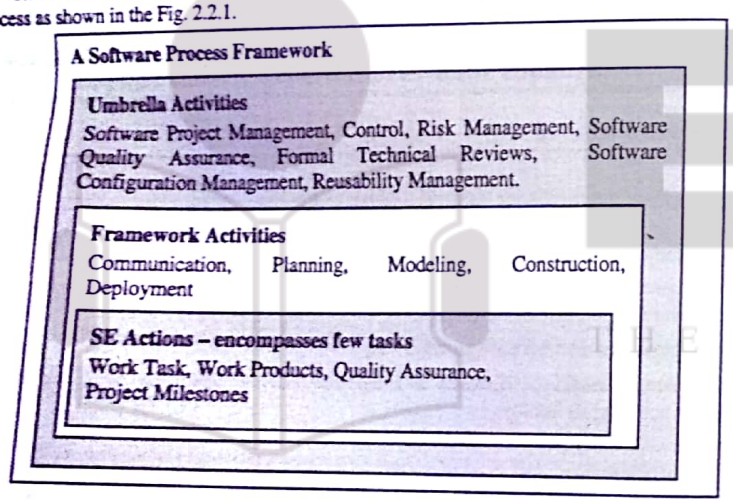


Fig. 2.2.1 : A software process framework

Again, each *framework activity* contains a set of *software engineering activities* which is a collection of tasks that develops a major software product.

2.2.1 Framework Activities

First, we take a look at the generic process framework activities that are must for the development of any software project :

☞ **Communication :**

The main cause of communication is requirement gathering. This activity establishes sufficient interaction or collaboration between the developer and the customer for gathering the requirements and knowing the expectations of the customer.

Example : We can explain the **work task** regarding the **communication activity** of a simple project as listed below :

- Making the list of the end-users, software engineers and support people for the project.
- Inviting all of them for an informal meeting.
- Ask each end-user to make a list of features and functions required.
- Discuss these requirements and prepare a final list.
- Arrange the requirements according to their priority.
- Note the areas of uncertainty.

☞ **Planning :**

This activity defines the software development process to be conducted. It describes all the needed technical tasks, possible risks, the resources that are required, the work product to be produced and the schedule to workout the whole process. This activity plans the work, identifies the resources, tasks and sets the schedule.

☞ **Modeling :**

This activity creates a model (blueprint) which clearly describes the software requirements and the design that will achieve these requirements. This is helpful to both customer and the developer respectively, to understand what he wants from the software and how he can develop it. Modeling is composed of two main activities-*analysis* (requirements gathering, elaboration, negotiation, specification and validation) and *design* (data design, interface design and each module level design).

☞ **Construction :**

This activity includes code generation either manually or using automated tools and then testing the code to correct the errors if any.

☞ **Deployment :**

The software (as a complete product or in a partial stage) is delivered to the customer who then checks the product and provides feedback on evaluation.

The framework activities are applied on every project but the degree of tasks depend on the :

- Type of the project
- Characteristics of the project
- Agreement of the project team on common views.

2.2.2 Process Iteration Activities

The above framework activities discussed in Section 2.2.1 occur in an organized pattern with respect to sequence and time. This work flow pattern of the activities is termed as ***'Process Flow'***.

1. **Linear Process Flow :** Executes the five framework activities in a sequence starting with 'communication' and ending with 'deployment'.
2. **Iterative Process Flow :** Repeats one or more of the five framework activities before proceeding to the next.
3. **Evolutionary process Flow :** Executes the five framework activities in a 'circular/cyclic' manner.
4. **Parallel Process Flow :** At a time, executes one or more activities i.e. one or more of the five framework activities are executed in parallel with the other. Say, modelling of one module is executed parallel to the construction of another module.

Fig. 2.2.2 : Linear Process Flow

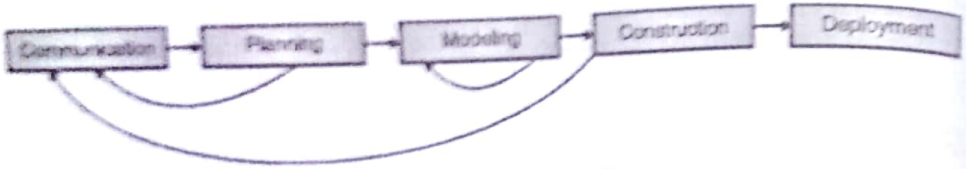


Fig. 2.2.3 : Iterative Process Flow

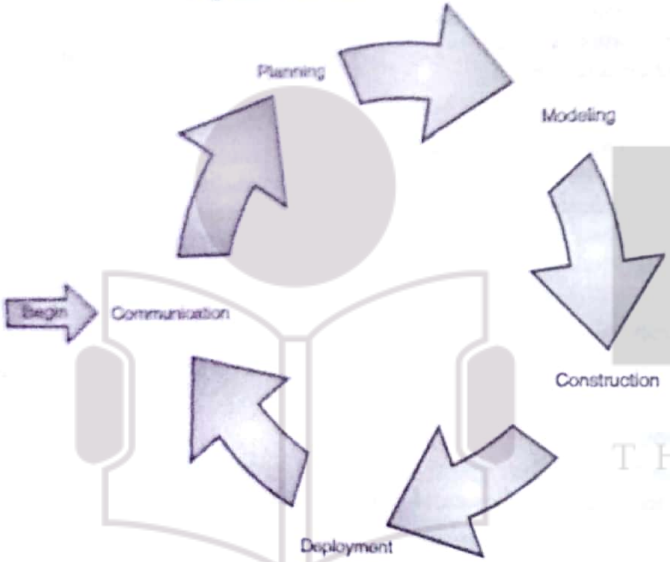


Fig. 2.2.4 : Evolutionary Process Flow

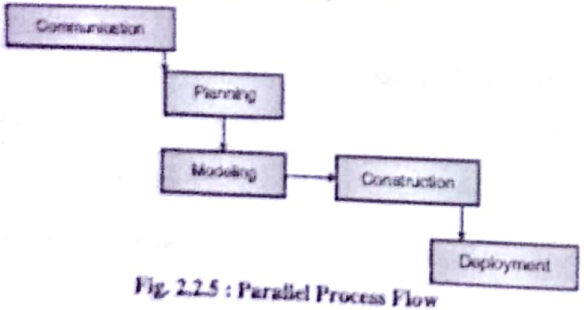


Fig. 2.2.5 : Parallel Process Flow

2.2.3 Umbrella Activities

*Now, we look at the **Umbrella activities** that are applicable throughout the software process :*

- **Software Project Tracking and Control :** Software team assesses the progress of the project plan time to time and takes necessary action to maintain the schedule. Thus, software team tracks and controls the project schedule.
- **Risk Management :** Software team assesses the risks that may affect the outcome of the project or say the quality of the product.
- **Software Quality Assurance :** Software team defines and conducts the activities needed to preserve the quality of the software product.
- **Formal Technical Reviews :** Software team assesses the technical efforts to find and remove the errors before they are forwarded to the next action.
- **Measurement :** Just the coincidence is the four P's of Software Engineering : Project (the task at hand), Process (the manner it is done), Product (the object produced) and People (by whom it is done). Software team collects all the project, process and product measures so that it can be used in combination with all other framework and umbrella activities.
- **Software Configuration Management :** It is about managing the changes and their version throughout the software process.
- **Reusability Management :** defining the criteria for work product reuse and establishes mechanisms to achieve reusable components.
- **Work Product Preparation and Production :** proper planning so as to create work products such as models, documents, logs, forms and lists.

Syllabus Topic : The Waterfall Model

2.3 Waterfall Model

Q. Explain waterfall model in brief.

This model was proposed by the Winston Royce. It is also called as a ***classic life cycle.*** It suggests systematic sequential approach for software development. It is oldest paradigm for software engineering.

It begins with software requirement and customer specification and progress through planning and testing.

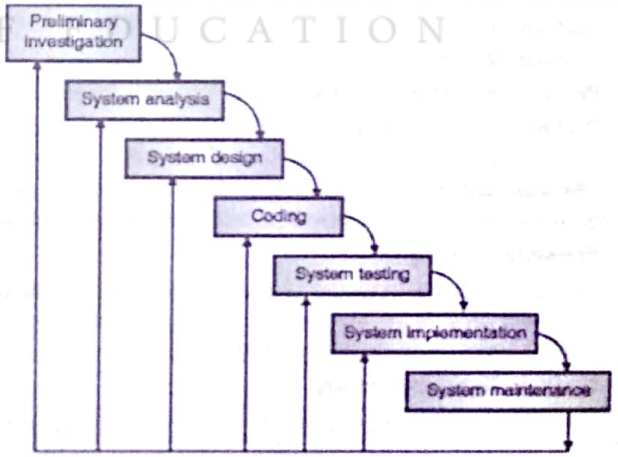


Fig. 2.3.1 : Waterfall model

Step 1 : Preliminary Investigation

Preliminary investigation means total inspection of the existing system i.e. clear understanding of the system.

Its basic task is to find out real problem of the system with causes and complexity of the problem. Its secondary but very important task is to find out all possible solutions to solve that problem and according to that which solution is feasible for the system in terms of technology, cost, operational. Its last task is to mention all benefits can be expected after problem is solved.

So this phase is divided into three main goals as follows:

1. Problem identification
2. Possible and feasible problem solution i.e. Feasibility study.
3. Expected benefits after the problems are solved.

Problem Identification

It requires to completely investigate the environment of the system. Generally it requires studying two environments - Internal environment and external environment, which are listed below :

Sr. No.	Internal Environment	External Environment
1.	Company Management	Customers
2.	Employees of all departments	Management consultant
3.	Internal auditors	External auditors
4.	Data Processing department	Government Policies
5.	Financial Reports	Competitions

There are normally seven types of problems encountered in the system :

1. **Problem of Reliability :** If system may not work properly or same procedures give different (i.e. unreliable) result.
2. **Problem of Validity :** Reports contain misleading information.
3. **Problem of Economy :** System is costly to maintain.
4. **Problem of Accuracy :** Reports have many errors.
5. **Problem of Timeliness :** Every work requires large time.
6. **Problem of Capacity :** Capacity of the system is inadequate.
7. **Problem of Throughput :** System does not produce expected results, or we can say system has more capacity but it accomplishes very less work as compared to capacity.

The main advantage of waterfall model is, it exactly pin points the problem. So it is very useful in setting all goals of the system as well as used to decide system boundaries.

Feasibility Study

Feasibility study is essential to evaluate cost and benefit of the proposed system. This is very important step because on the basis of this; system decision is taken on whether to proceed or to postpone the project or to cancel the project.

Need of Feasibility study

Q. What is the need of feasibility study? Explain its types.

1. *It determines the potential of existing system.*
2. *It finds or defines all problems of existing system.*
3. *It determines all goals of the system.*
4. *It finds all possible solutions of the problems of existing system. We can call it as proposed system.*
5. *It finds technology required to solve these problems.*

6. It determines most suitable solution
7. It determines the required hardware and software.
8. It does the cost estimation in terms of cost of hardware required, software required, designing new system, implementation and training, proposed maintenance cost.
9. It avoids costly repairs, crash implementation of new system.
10. It chooses such system which is easy for customer to understand and use so that no special training is required to train the customer. It may give some training to employees of the system.

Method-Steering committee

This committee conducts detailed study. This committee first studies existing system and identifies all problems and looks into three types of feasibility study. Those are given in Fig. C2.1:

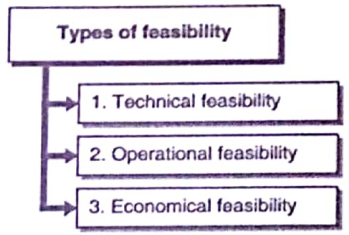


Fig. C2.1 : Types of feasibility

1. Technical feasibility

The committee first finds out *technical feasibility of the existing system.* It involves following steps :

1. It determines available hardware.
2. It determines available computer with configuration.
3. It determines available software.
4. It determines operating time of available system that is computer, hardware software.

After that it finds out *technical feasibility required for the proposed system.* It involves following steps :

1. It mentions new hardware requirements of proposed system.
2. It mentions computer with new configuration requirement of proposed system.
3. It mentions new software requirements of proposed system.
4. It mentions new operating time of available system that is computer, hardware, software.
5. It mentions old as well as new facilities which will be provided by the proposed system.
6. It mentions all benefits of the system.

2. Operational feasibility

It is also called as behavioural feasibility. It finds out whether the new technology or proposed system will be suitable using three type of aspects; that are human, organizational and political aspects. It involves following steps :

1. It finds the ease of operation of proposed system compared to existing system.
2. It finds if the users of the system require any extra training ?
3. It finds out whether the user or customer of the system requires extra training or not ?
4. If it requires any retraining then it is accepted by user as well as customer or not ?
5. It finds if any job reconstruction is required or not ?
6. It finds if this reconstruction of the job is accepted in organization ?
7. It also finds if it is acceptable then what should be the skill sets of that job.
8. Watches the feelings of the customers as well as users.
9. It should provide right and accurate information to user or customers at right place as well as at right time.

→ 3. Economical feasibility

Here, steering committee finds total cost and all benefits as well as expected savings of the proposed system.

There are two types of costs – one time cost and recurring costs.

One time cost involves :

1. Feasibility study cost.
2. Cost required to convert existing system into proposed system.
3. Cost of hardware's, OS, application software.
4. Technical experts consulting costs.
5. Cost of training.
6. Cost of Documentation

Recurring cost involves following :

1. Cost involves in purchase or rental of equipment.
2. Cost of phones and mobiles Communication equipment.
3. Cost of Personnel search, hiring, staffing.
4. Cost of Salaries of employee's.
5. Cost of supplier's.
6. Cost of maintenance of equipment.

Step 2 : System Analysis

This phase of the water fall model is nothing but complete understanding of all important facts of the business using preliminary investigation. It involves following steps :

1. It is nothing but study of all components of the system as well as inter relation between all components of the system and relation between components and environment.
2. It determines what is to be done in the organization.
3. It finds the procedures of how to do that.
4. Which is input data ?
5. What is the procedure through which inputs are to be converted into output ?
6. When transaction should occur on the data.
7. When problem arises, determine the solutions to solve it and what are the reasons of those problems.

Objectives of System Analysis: The main roles of the system analysis are :

1. Define the system.
2. Divide the system into smaller parts.
3. Finds all nature, function and inter-relationship of various parts of the systems.
4. If the system is not analyzed properly then there may be problem in the Preliminary investigation phase.

Step 3 : System Design

The main objective of this phase of waterfall model is to design proposed system using all information collected from preliminary investigation and directed by the system analyst. This is very challenging phase. It includes following steps :

1. Design of all types of inputs of proposed system.
2. Design of all types of outputs of proposed system.

3. Design of the procedures which convert input to output.
4. Design of the flow of information.
5. Design of the information which is required to store within a files and data bases Volumes.
6. Design of collection of inputs using forms (Manual forms).
7. Design in terms of program specification i.e. logical design.
8. It determines the hardware cost, hardware capability.
9. It determines the speed of software.
10. It determines error rates, and other performance characteristics are also specified.
11. It also considers the changes to be made in the organizational structure of the firm in design.
12. This phase also designs standards for testing, documentation.

Generally traditional tools are used for the designing of the procedures that are as follows :

- Flowcharts
- Algorithms
- IPO (i.e. Input Processing and output) and HIPO (Hierarchy of IPO) charts
- Decision tables
- Data Flow diagrams

1. If system design phase is facing problem during the design then first go back to the system analysis phase and redesign the system but if problem is not solved then go for preliminary investigation.
2. If System design phase produces all expected results then it goes to next phase.

Step 4 : Coding

This phase is just implementing the design in to programming language that means it actually develops the proposed system. It involves the following steps :

1. It first of all, finds out the best suitable programming language that is suitable for the design as well as also suitable in the organization.
2. It accepts design and break system modules into smaller programs.
3. It develops or writes program for each part in selected programming language.
4. Prepares documentation that means add necessary comment lines wherever necessary within a program.
5. Now it combines all small programs together and builds one big program.
6. If any problem occurs during the coding phase then waterfall model tries to solve it by repeating system design phase:
   a) If that problem does not get solved then waterfall model repeats system analysis phase and system design phase.
   b) If that problem does not get solved then waterfall model repeats from first phase preliminary phase through system analysis phase and system design phase.
7. If coding phase produces all expected result then it goes to next phase.

Step 5 : System Testing

This phase includes the testing of the code or programs developed by the coding phase. This includes following steps :

1. First of all, it finds out all possible expected results (i.e. output data) for the set of input data.
2. It also checks the validity of the input data as well as checks expected output data.
3. It finds out all wrong results and immediately tries to correct it by repeating coding phase.
4. It finds the speed of functions using special codes.
5. It determines whether each program can perform the intended tasks or not?

6. It checks result by test data.
7. It checks the logic of the individual programs.
8. It checks interfaces between various programs.
9. It checks quality of code in terms of speed and space.
10. It checks whether system has produced correct and desired results which lead to designated goals.
11. If testing does not produce expected result then waterfall model tries to solve it by repeating system design phase and coding.
    a) If that problem does not get solved then waterfall model repeats system analysis phase through system design phase and coding.
    b) If that problem does not get solved then waterfall model repeats from first phase i.e. preliminary phase through system analysis phase, system design phase and coding.
12. If testing phase produces all expected results then it goes to next phase.

Step 6 : System Implementation

System Implementation is not creative process but it is some what difficult task. This phase has two parts - *implementation* and *evaluation* of the system.

☞ Implementation

There are two ways of implementation. Those are as follows :

1. Implement proposed system with existing old system and find out performance of the both systems and slowly replace new system with older one.
2. Totally replace old system with proposed new system.

Risk factor of second type of implementation is more as compare to first one. Second step needs strict evaluation.

Both types of implementation consist of following steps :

1. It prepares site for new proposed system.
2. It installs required hardware within a system.
3. It installs required software in a system.
4. It installs a developed code i.e. programs in a system.
5. It prepares training program for user of the proposed system as well as customers of the system.
6. It prepares user manual which includes all the steps which give guidance to user.
7. It gives training to all types of user of proposed system.
8. Observe the system when users of system are using it.
9. If users are facing any problems regarding the new system, it tries to find out exact phase from where root cause of the problem starts, and accordingly starts waterfall model.

☞ *Evaluation*

Evaluation is nothing but feed back for the system. It is very essential check point of the system which is the process of verifying the capability of a system. It continuously evaluates and checks whether proposed system is meeting the objectives or not. It includes :

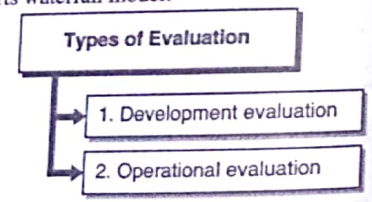


Fig. C2.2 : Types of Evaluation

➜ ***1. Development evaluation***

- *It checks whether the system is developed within time.*
- *It checks whether the system is developed within the budget.*

- System is assed by development methods and tools.

➜ **2. Operational evaluation**

- It checks response time of proposed system.
- It checks whether it is really easy to use or not?
- It checks accuracy of computations (It is seen in testing also).
- It checks storage capacity.
- It checks reliability.
- It checks functioning of the existing system.
- Collects necessary feedback from users.
- It finds all benefits of the proposed system.
- Collects information of attitude of different persons regarding proposed system.
- It evaluates cost, time and effort taken for the overall project.

Step 7 : System Maintenance

Maintenance is the process, in which it finds out essential changes (i.e. new trends) of the market or business or to correct some errors and tries to implement it in the existing system. There are usually three types of maintenance, that are :

1. **Correction :** Some times, proposed system has few types of errors; and it is the duty of software engineer to correct it as soon as it is encountered by the user. Generally there are four types of errors, that are as follows :
   - Minor changes in the processing logic.
   - Errors detected during the processing.
   - Revisions of the formats for data inputs.
   - Revisions of the formats of the reports.

   These errors can be corrected by repeating waterfall model from coding phase through testing, implementation and maintenance.
2. **Adaptation :** Some times, our proposed system is executable on Windows environment, but somebody wants to run it in LINUX environment, or some other operating system. Then we are required to design our proposed system from third phase that is from System Design phase.

   This is actually error free system. It is good so other people also want same system in
3. **Enhancement :** Because of new technology and business competition, organization needs to imply or to add new functions or additional capabilities to the proposed system.

   After some time, people think that some techniques may be used in the system so some additional features can also be added into it. Sometimes, new hardware is required to add some extra features.

   For enhancement it may repeat whole system or may repeat it from design phase or some times from coding phase.

☞ Advantages of waterfall model

1. It defines very first software development process.
2. The product of waterfall model always defines all constraints of the organization.
3. It always produces a good quality product in terms of space and time.

☞ Disadvantages of waterfall model

There are some disadvantages of the waterfall models that are as follows :

1. Real products rarely follow this sequential flow.

2. Because of iteration, changes can cause confusion as the project team proceeds.
3. It is very difficult for customer to state all the requirements in one time.
4. Many projects face this uncertainty at beginning only , so it is very difficult to design next phases.
5. Time span required for each phase could not be specified.
6. Naturally project requires more time.
7. Project becomes lengthy also.
8. Customer should have patience.

To overcome these drawbacks of waterfall model, a new model was designed known as ***'Incremental process model'***. Incremental process model has two process models 1) Incremental Model and 2) Rapid Application Development model.

Syllabus Topic : Incremental Process Models

2.4 Incremental Process Models

Method

- As soon as customer comes to software engineer and gives requisition of new project to software engineer; he starts to collect information from user to start preliminary investigation.
- He immediately increments towards second phase and third phase and starts to analyze and design the system.
- He starts to convert design into coding i.e. as time or calendar grows, some functions of software are also progressed towards completion.
- When testing of first requirement goes towards completion stage, customer comes with second requirement. Immediately he starts to collect all essential information for second increment i.e. preliminary investigation of the second requirement in parallel.
- He starts implementing first requirement i.e. module 1 as well as starts designing the second requirement i.e. module 2.
- Maintenance phase of first module is continuing then simultaneously coding of second module is starting as well as customer may ask for third requirement and he increments for the third incremental model.

This is explained in following graph ;

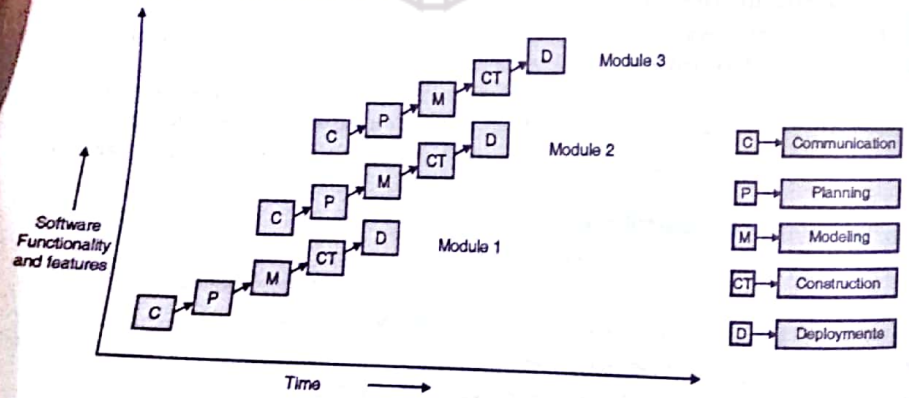


Fig. 2.4.1 : Incremental model

Example of Incremental Model

The software word processor is designed using incremental model. It is designed as follows :

1. Customer comes to software engineer and gives requisition of new project of a **word processor** to software engineer.
2. He starts to collect information from customer. Customer tells him to design file management system with all editing functions and document production functions.
3. Immediately software project team starts preliminary investigation to decide the plan.
4. Software engineer immediately increments towards second phase and third phase and starts to analyze and design the system. When third phase of first increment leads towards completion state then customer comes with second requirement i.e. about more sophisticated editing capabilities like copy, paste, and find capabilities. He starts preliminary investigation of the second increment in parallel.

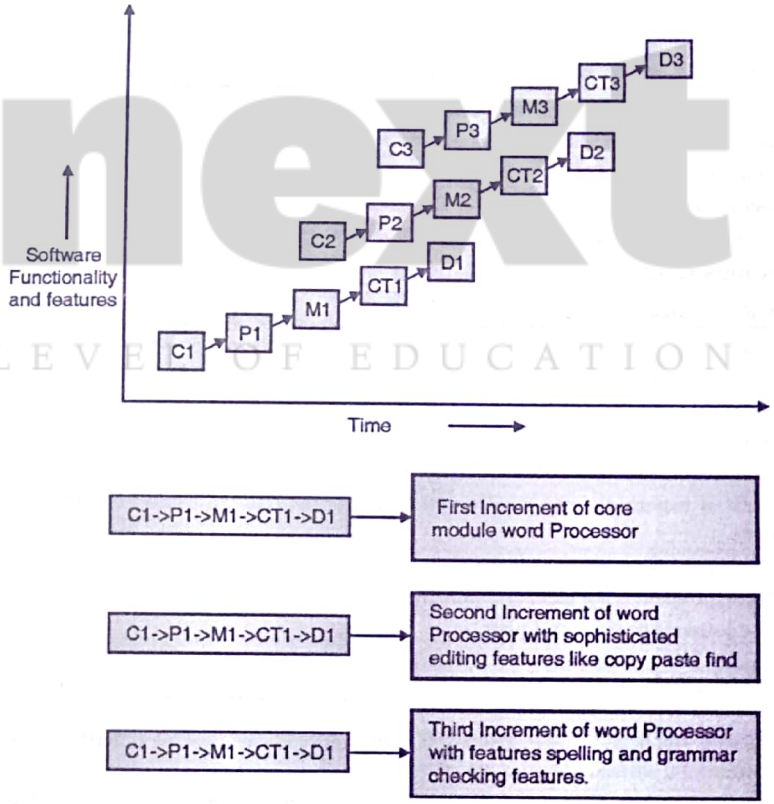


Fig. 2.4.2 : Example of incremental model

5. He starts to convert design into coding of first increments as well as he designs plan and objectives of the second increment for these sophisticated editing capabilities.
6. He starts implementing first requirement i.e. core module as well as it starts design of second requirement i.e. second module.
7. Maintenance phase of first module is continuing then simultaneously coding of second module is starting as well as customer may ask for third requirement that is spelling and grammar checking feature and it starts for the third incremental model.

This is explained in Fig. 2.4.2.

Method used in Incremental model

1. Generally in incremental model, the first increment is a core product which includes maximum basic requirements.
2. Second increments and third increments include all supplementary features (known and unknown features) which increase additional features with more functionality in a product.
3. This process is repeated till the completion of the final product.

Advantages of Incremental model

Q. List the advantages of Incremental model.

1. Preliminary investigation time is very small or reduced.
2. It requires very less time as compare to waterfall model
3. It requires less number of staff to develop the system.
4. Customer is satisfied because of quick development of the new system.
5. If system has many increments its functionality also increases; thus, the product has many features.

Disadvantages of the Incremental model

Q. List the drawbacks of Incremental model.

1. All the tasks are not decided in first phase so there may be some problem with designing phase. That means, overall design of the system is not so good.
2. It may produce software, which requires large space in memory.
3. Some times, speed of software is also slow.
4. Delivery date of the product could not be decided.
5. New levels of increments may require new hardware because old hardware is involved with old increments.
6. Some times, it produces same code in different module because of partial functionality.
7. Some times, problem is not identified properly.
8. Quality of software or product is poor.

2.5 RAD Model

RAD is Rapid Application Development. It is high speed adaptation of waterfall model. It is the example of incremental software process model.

The main disadvantage of incremental model is - the quality of product is very poor as well as it also requires more time to develop. So this model is only designed to produce a good quality product in very short duration of time.

Method

1. It also adapts generic framework activities like waterfall model that means five phases.
2. This allots sufficient time for study of the existing system.
3. It accepts sufficient number of staff for development of the system.
4. Staff is distributed into various teams (i.e. senior programmer junior programmer ,team leader, project leader etc).
5. Communication phase is used to understand business problem as well as it tries to identify all type of problems and gather all type of information.
6. Planning is essential because it divides work into manageable different parts and distributed among the various teams.
7. Modeling will be accomplished by different teams simultaneously. It includes three different phases. Business modeling, Data modeling and process modeling.
   - Business modeling collects all essential information about the product from market view or from the business point of view.
   - Data modeling shows the flow of data using some techniques like DFD.
   - Process modeling performs each process which handles a flow of data. It decides activities performed in each process.

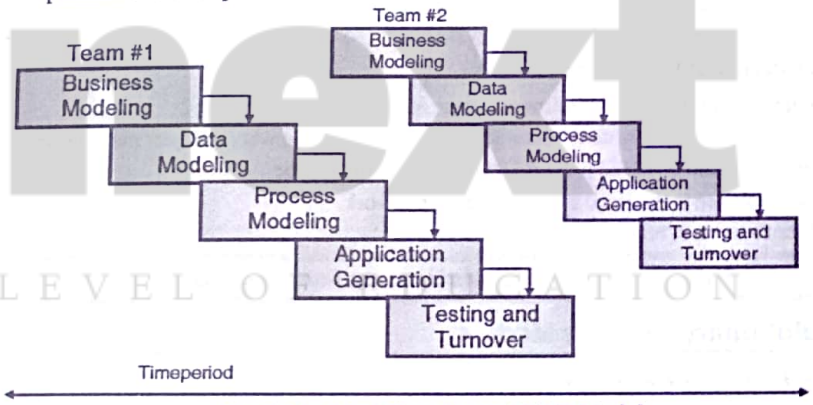


Fig. 2.5.1 : Modeling phase of RAD Model

   - Application Generation is about using a set of automated tools to facilitate the construction of the software say for example, the 4th GL techniques.
   - Testing and Turn over : Many of the components that will be used in the development of the proposed system might have already been tested since RAD focuses on reuse which reduces overall testing time. It is needed to concentrate on testing the new components.

Once modeling activity of each team is over, it immediately starts construction phase.

8. In construction phase, each team develops the code of our product as well as performs testing. It reuses old software components and develops new functions which are called as automatic code generation.
9. Deployment collects each code from different team and clubs them into single project implement it and if required then perform iteration among the previous phases.

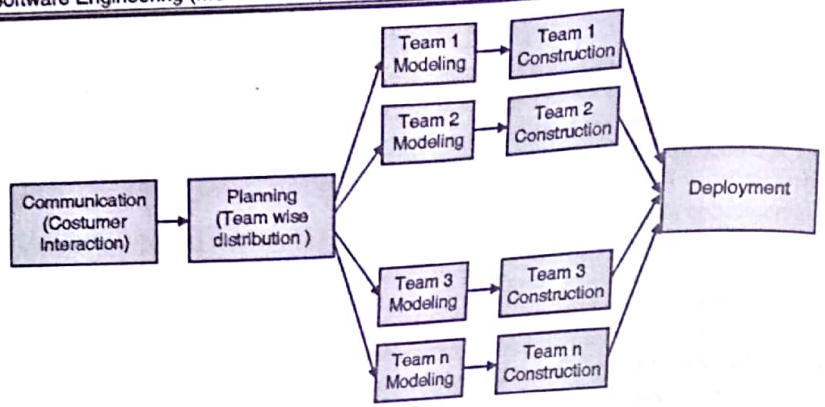


Fig. 2.5.2 : RAD model

Advantages of RAD model

1. It requires very less time to develop the product.
2. Planning and Design is performed before construction so it is not lengthy.
3. Product may be produced before its delivery date.

Disadvantages of RAD model

1. Large projects require sufficient or more human resources.
2. If developers and customers do not interact with each other then whole project may elapse.
3. If system is to be modularized properly then RAD is problematic.
4. If high performance is the Issue then RAD does not work.
5. It also doesn't work in high technical risk.

Syllabus Topic : Evolutionary Process Models

2.6 Evolutionary Process Models

The main drawbacks of Incremental process model are :

1. Overall design of the system is not so good.
2. It occupies large space in memory.
3. Some time speed of software is also slow.
4. Delivery date of the product is not decide.
5. Some times, problem is not identified properly.
6. Quality of software or product is poor.
7. Lack of interaction between developers and customers.
8. Poor modularization.
9. Doesn't produce high performance.
10. It is also not work in high technical risk.
11. Not suitable for complex product

These drawbacks are overcome in Evolutionary process model.

Main objectives of Evolutionary Process model are :

1. It should be suitable for complex system.
2. As time grows business also changes and product requirement may change during its development or construction phase.
3. It makes straight line path between customer requirement, business requirement and unrealistic product.

4. Evolutionary products should be iterative.

Though Evolutionary process model provides these advantages, it also has some issues.

Problems faced while using Evolutionary process models

1. Business competition makes product unrealistic
2. Tight market deadlines.
3. During development, product extension may be possible.
4. Software engineers must be flexible to requirements of customer and business changes.

2.6.1 Prototyping

This is the method or model of evolutionary process model. When customer tells his requirements, he is not aware about the development process, detailed input processing and also developer may be unsure of efficiency of the algorithm and adaptability of operating system, then prototype is the first step which is a best approach. This is explained in Fig. 2.6.1.

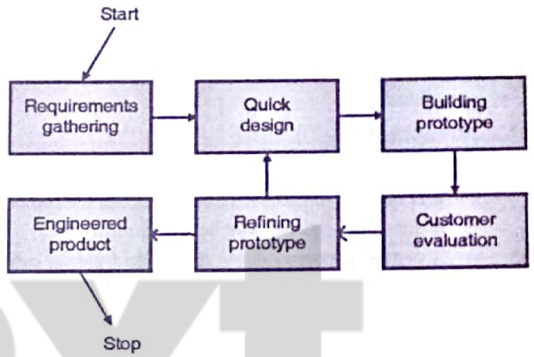


Fig. 2.6.1 : Prototyping model

Method

1. It begins with ***requirement gathering*** phase. Here customer and software engineer meet and define the overall objective of the product, It includes
   - Problem identification
   - Requirement analysis
   - Outline areas where further definition is mandatory

   According to overall objective of the product, software engineer makes planning of the design quickly.
2. ***Quick design*** focuses on the representation of human interface i.e. those aspects that are visible to the end user and customer. It uses different tools for design. Then, the analysts estimate a prototyping cost and inform about it to the management.
3. ***Building Prototype*** : Depending upon the quick design, construction of prototype is done that includes the software programs having the following capabilities :
   - accept input,
   - validate data,
   - perform calculations,
   - interact with files
   - produce outputs
   - Application generators

   Such a prototype can be built using several tools such as :

Screen generators :

- Input data screens with data validation are prepared.
- Meaningful prompt messages with each data entry
- Output screens are prepared.

- Table should have column headings and row heading.
- Labels, Messages, Colours, Fonts etc are decided.

☞ **Report generators :**

It shows output according to users requirement where reports are made from records extracted from data base.

4. ***Customer Evaluation*** **:** The built prototype is then evaluated by the customers or end-users to find if any changes are required. If so, then the requirements of the proposed software are refined.
5. ***Refining Prototype*** : The prototype is again refined based upon the feedback of the end-users about what they need and what they don't expect. This process is repeated till both the end users and the developers feel that all the requirements are fulfilled in the software and thus there is no need of refinement.
6. ***Engineered Prototype*** **:** This completed product is then given to the customer as per the specification.

☞ **Advantages**

- It is simple and an iterative process.
- It is revised to satisfy the needs of the customer.
- It does not require more cost to build.
- It can be prepared using pencil and paper, or computer software like screen generators, report generators and application generators.
- It saves time of development.
- It develops the product through ongoing communication with the user.
- So easily finds user friendly services and non user friendly services.
- Readymade tools are used, to development of code.
- Product is to be delivered within proper time.
- It provides training to user.
- Prototypes are also used for testing.

☞ **Disadvantages**

- It does not contain all features or perform all the necessary functions of the final system.
- Customer/user evaluates the module/prototype and they suggest addition and modifications so it may become lengthy.
- So may produce poor level quality product.
- Developer may compromise with operating system.
- It is not linear.
- It is not finding the risk of the project.

Q. State the difference between :Classic life cycle model and Prototyping model.

Table 2.6.1 : Comparison between Classic life cycle and Prototyping model

Sr. No.	Classic Life Cycle Model	Prototyping Model
1.	It is not an iterative process.	It is an evolutionary i.e. iterative process model
2.	During development, mostly product extension or addition of new requirements is not encouraged	During development, product extension may be possible.

Sr. No.	Classic Life Cycle Model	Prototyping Model
3.	Testing is not conducted from the initial phases of the SDLC – therefore when errors are detected after the coding phase it costs a lot for refining the product.	Building and refining of product is involved before the actual engineering process – therefore, much cost is not involved
4.	Included Manual coding	Uses readymade tools such as CASE tools for coding
5.	Communication with the users is done at the start i.e. while requirement gathering and then while testing phase. No ongoing communication with the users is encouraged	develops the product through ongoing communication with the user

2.6.2 The Spiral Model

Q. What does the radius imply in Spiral model ?

The Spiral model is also an evolutionary software process model that couples the iterative nature of prototyping. It is proposed by Boehm. The main objectives of spiral model are ;

- It provides controlled and systematic aspects of the linear sequential model.
- It uses potential of rapid development of incremental versions of the software.
- It finds all risks in the project.
- It finds future risks also.
- It is explained in following diagram.

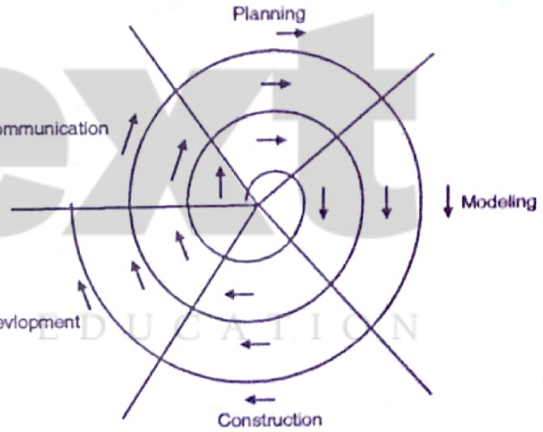


Fig. 2.6.2 : Basic spiral model

☞ **Method**

Requirement Gathering includes;

- Interaction with customer and user.
- Problem identification regarding existing system.
- Deciding product specification.
- Deciding objectives of the system.

Planning decides the project plan. It includes

- Cost measurement of the system.
- Deciding schedule of the system.
- Collecting feedback from the user.
- Adjusted and planned number of iterations required to complete software.

Risk Analysis task is involved in each of the iterations of the project. It includes;

- Identification of risk areas
- Identification of technical as well as managerial risks
- Measuring cost related risk
- Efforts are taken to resolve the risk

Engineering phase immediately designs the prototype of the proposed system depending upon the objectives decided in the planning phase. It also converts design into coding and does the needful testing. It then shifts to *deployment phase* which includes :

- Implementing the developed system
- Do the Verification
- Gives training to users
- Collect feedback

Customer Evaluation : The product is then evaluated by the customers or end-users to find if any changes are required. If so, then the above phases are repeated again in loops.

Q. State the difference between : Waterfall Model and Spiral Model.

Table 2.6.2 : Comparison between Waterfall and Spiral Model

Sr. No.	Waterfall Model	Spiral Model
1.	Process flows from top to bottom like a flow of water from a hill to ground. Flow of water can't be reversed - similarly any new changes cannot be incorporated in the middle of the project development.	Best suitable for projects associated with risks.
2.	Process goes to the next phase only after the completion of the previous phase. Here, end user feedback is not taken into consideration for any change in SRS. This will result in restarting the work from beginning.	Each and every step goes through testing which makes it easy to recover any error and fix it then and there itself. In this model we don't have to start work from beginning.
3.	Purely a pre planned /strategic in nature.	Used to build a product which doesn't have adequate requirement gathering.
4.	Stresses more on requirement gathering.	focuses more on risk analysis which is not much considered in waterfall model.
5.	Customer feedback is not considered at every step of project development.	Customer feedback is considered at every step of project development.

☞ ***Advantages***

- *It develops large scale systems and software.*
- *Better understanding of developers and customers.*
- *It performs risk analysis from the point of view of technical aspects as well as managerial aspects.*
- *It follows systematic and stepwise approach of SDLC.*
- *Wherever needed, it uses prototyping approach during development.*

☞ ***Disadvantages***

- *It is not suitable for fixed budget development.*
- *It is not panacea.*

- It demands for expertise for risk assessment.
- If major risk is not covered then problem is never solved.

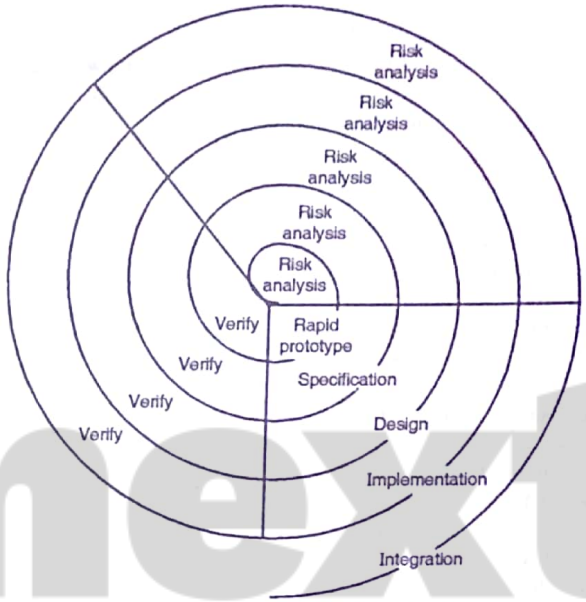


Fig. 2.6.3 : Spiral model used in industries

Now days, spiral view is to do risk analysis and prototyping that means to follow traditional framework then go for verification. If necessary goes for second iteration. This is well explained in Fig. 2.6.3.

Syllabus Topic : Concurrent Models

2.7 Concurrent Model

It is also called as a concurrent engineering. It also represents systematic framework activities with associated states. It is explained as follows;

- Every phase is associated with states.
- Every state is having some meaning full names.
- It is applicable for all types of the software development.
- It shows us exact state of the project.
- It supports parallel development of the project.
- It defines series of event transitions from state to state for each development activity.
- It defines network of activities, actions and tasks.
- It finds which activities or actions or tasks can be performed simultaneously.

☞ **Example of Modeling phase in Concurrent process model**

- It consists of 6 states that are ;

1. **Under development state :** When communication is over then it starts modeling and its state is Under development state.
2. **Awaiting changes :** It always collects requirements from the user. As soon as user enters requirements then the current state of the project is Awaiting changes state.
3. **Under revision :** It accepts changes from customer and starts working on it by revising the system. At this time, state of the system will be under revision state.
4. **Under review :** It collects necessary information from communication phase and Awaiting changes phase and study them.
5. **Base lined :** It design the system
6. **Done :** Once design is completed and there is no new requirement from user then the state of modeling is done that means, we can start construction phase.

- When communication is over, it starts modeling phase.
- It collects information of user from awaiting changes state.
- Then it revises the system and goes for under review state.
- Now prepare base line of design.
- Once design is prepared and there are no changes from customer then current state of modeling phase is done that means we can continue for new state that is construction state.

This is explained in Fig. 2.7.1.

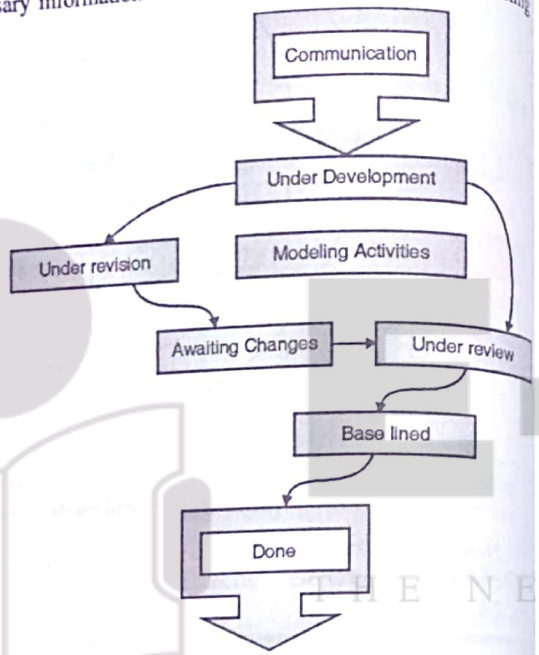


Fig. 2.7.1 : Concurrent development model

Syllabus Topic : Component-Based Development

2.8 Component-Based Development

- *Component-based development is about the **assembly of components**. It enables **Reuse**.*
- *Component-based development is a set of pre-build, standardized software components that are made available to fit in a specific architectural style for some or the other application domain.*
- *The application is then formed by assembling these available components, instead of assembling "discrete parts" of a conventional programming language.*

***Example :** when we purchase a "stereo system", we see that each component has been designed to fit in a specific architectural style - connections are standardized, communication protocol has been pre-established. Also, assembling of the components in this stereo system is easy because you don't have to build the system from scratch; instead you have all the components ready and you just have to connect them properly to each other. This is what 'CBSE' is all about.*

Design Principles

- Component-based development is *rapid assembly* and maintenance of component-based systems; where components have well-defined properties.
- A software component is a logically cohesive, loosely coupled module that denotes a single abstraction and can be ***reused*** in the system development.
- These components are ***independent*** of each other i.e. they do not interfere with each other because;
  - Every Component implementation is hidden from the other.
  - Communication between components is through well-defined interfaces.
- A software component is a service provider : The services offered by the component are made available through an interface and all component interactions take place through that interface.
- A software component is deployable only if it is self-contained and operates as a stand-alone entity on some component platform that implements the component model.

Advantages

- Increases competitiveness by;
  - Reducing the cost of software development.
  - Increasing software productivity.
- Limited human talent by

  Increasing software / person - It is possible by reuse of existing solutions, rather than invent them.
- The reuse of a component requires less time than the development of a new component. Therefore, systems can be built faster using CBSE process.

Component-based development Process

- Outlining the system requirements.
- Searching for components.
- Modifying the requirements according to available functionality in the components.
- Searching once again to find if there are better components that may meet the revised requirements.

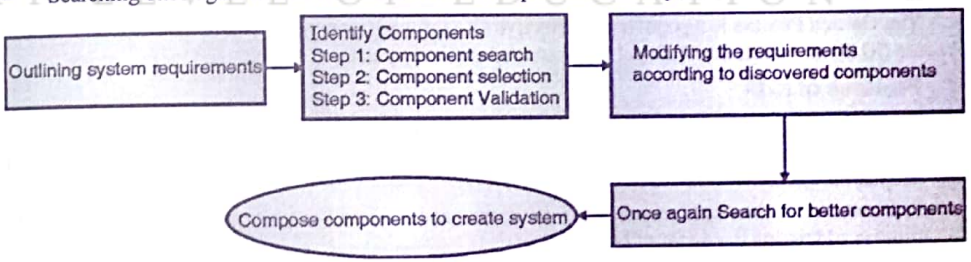


Fig. 2.8.1 : Component-based development process

Component Composition

- Composition is the process of assembling components to create a system.
- It involves integrating the components with each other and with the component infrastructure.
- Usually, 'glue code' is written to integrate the components.

Composition types

1. **Sequential composition :** The composed components are executed in a sequence.
2. **Hierarchical composition :** One component calls on the services of another.

3. **Additive composition :** The interfaces of two components are put together to create a new component.

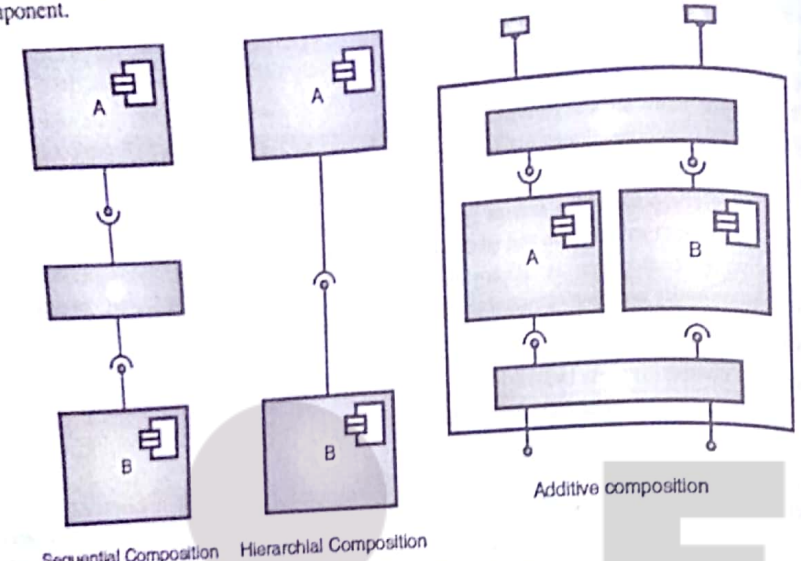


Fig. 2.8.2 : Composition types

Syllabus Topic : The Unified Process Phases

2.9 The Unified Process Phases

Q. State the purpose, advantages and drawbacks of RUP?

The Unified Process is a popular and effective OO software development process. Rational Unified Process (RUP) is modified at Rational Software and is widely practiced and adopted by industries.

☞ Features of RUP

- The most important feature/idea in RUP is *Iterative Development*. Iterative Development is sequentially expanding and refining a system through multiple iterations, using feedback and adaptation.
- RUP can be a lightweight process addressing the needs of small projects - to more comprehensive process addressing the needs of large projects.
- RUP does early and continuous documentation of the most urgent and the most probable risks by proper planning and keeping follow up. This helps in mitigating the risks at early phases of software development.
- RUP uses visualization methods like UML to build models to understand the complexity of the system.
- RUP uses *use-cases* as test cases which allows end-user documentation and helps in designing.

☞ RUP Phases

- ***RUP is divided into 4 phases of iterative development :***
- Each phase has iterations, each having the purpose of producing an executable piece of software. The duration of iteration may vary from phase to phase.

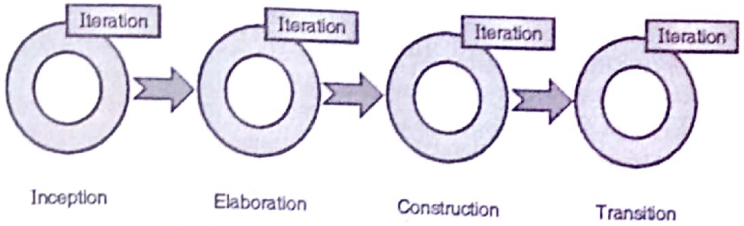


Fig. 2.9.1 : Four phases of Iterations in RUP

(I) Inception

Inception means start i.e. this is the point where the project is proposed.

Aim of Inception Phase : It constitutes of Business modeling i.e. Define the Problem, Scope of the System, and Initiate the Project.

☞ Purpose

- Define the problem : The objectives of the project are stated, so that the needs (requirements) of every stakeholder are considered.
- Define the scope of the system : The Scope and boundary conditions, acceptance criteria and some requirements are established. External entities (actors) with which the system will interact are identified and the nature of the interaction is defined on a high-level by identifying all use cases. It also includes identifying the business case i.e. identifying the success criteria, risk assessments and estimation of the resources needed and a phase plan showing dates of major milestones.
- Initiate the ***project*** : And then, you can start working on the project.

☞ Inception Outcome

- Vision document
- Initial use-case study (10%-20% complete)
- Initial business case
- Initial risk assessment
- Project plan
- Stakeholders decide whether to commence a full scale project or not.

(II) Elaboration

Elaborate means refinement (careful development).

Aim of Elaboration Phase : Detailed analysis of the problem resulting in the definition of an architectural foundation for the project. It constitutes of ***requirements, analysis*** and ***design*** phases.

☞ Purpose

- Analyze the problem
- Develop the project plan
- Remove the highest risk elements of the project.
- Gives you a *mile wide and inch deep* view i.e. little bit deeper view of the system.

Elaboration Outcome

- Use-case model is 80% complete.
- Additional requirements capturing the non functional requirements and requirements that are not associated with a specific use-case are identified.
- Description of the Software Architecture.
- An executable architectural prototype is developed.
- A revised risk list and revised business case is developed.
- A development plan for the whole project, including iterations and evaluation criteria for each iteration also specifying the process i.e. to be used.

(III) Construction

Construction means to build i.e. it is a manufacturing process.

Aim of Construction Phase : It constitutes of implementation i.e. detailed design and construction of source code.

Purpose

- Emphasize on managing resources and controlling operations to optimize costs, schedules and quality. This phase is broken into several iterations.
- All components and application features are developed and integrated into the product and tested.

Construction Outcome

- An executable product that is ready to put in the hands of the end users.
- The developed and ready to use software product
- A user manual
- Description of the current release.

(IV) Transition

Transition means delivery. The transition phase is the phase where the product is put in the hands of its end users.

Aim of Transition Phase : It constitutes of deployment i.e. delivery of the system to the user community. It involves issues of marketing, packaging, installing, configuring, supporting the user-community, making corrections, etc.

Purpose

- Deliver the software product to the user community.
- Issue new releases
- Perform beta-testing to validate new system against user expectations
- The system might run in parallel with a legacy system
- Roll-out the product to marketing, distribution, and sales team

Transition Outcome

- Achieving user self -supportability
- Achieving stakeholders' agreement that deployment is complete and consistent.

Six Best Practices of RUP

1. Develop iteratively

- Software must be developed in small increments and short iterations.
- An iterative process breaks a development cycle into a *sequence of 4 phases* each of which includes a *series of iterations.* (You can see this in Fig. 2.9.1.

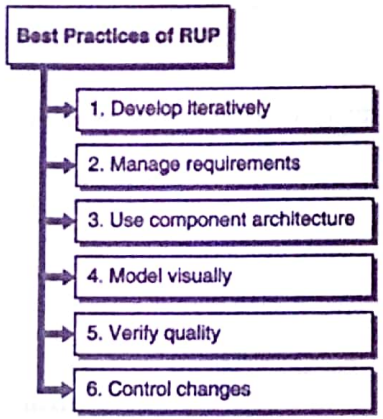


Fig. C2.3 : Best Practices of RUP

2. Manage requirements

- RUP allows accommodating requirement changes in system development strategy.
- Those requirements that change over time and those requirements that have greater impact on project goals are identified.
- It is a continuous process to identify requirements.
- Managing requirements include :
  - Elicit, organize (according to the priority), and document the required functionalities and constraints,
  - Evaluate the impact of changes and
  - Track and document the decisions.

3. Use component architecture

- The process focuses on early development and design of independent executable modules, prior to committing for full-scale development.
- Components that are most likely to change and components that can be re-used are identified and built.

4. Model visually

- Models must be built using visualization methods like UML, to understand the complexity of the system.
- This helps you to understand the different aspects of your software and see how the different elements of the system communicate with each other.
- Maintains uniformity between design and its implementation.
- Promotes unambiguous communication between developer and end user.

5. Verify quality

- Quality of the software is maintained by its frequent testing.
- Testing is done to remove defects at early stages, thus reducing the cost at later stages. In particular, high risk areas are tested more thoroughly.
- The software released at the end of every iteration, is tested and verified.
- Test cases are created based on use cases (and its scenario).
- Decisions are made on real test results.

6. Control changes

- Any changes to requirements must be managed and their effect on software should be tracked.
- All change control goes through the convener of the CCB (Change Control Board).

- Members of CCB can be representatives from different areas, say: test designer, project manager, system analyst or stakeholders.

Advantages of RUP

- RUP helps in addressing very early high risks areas.
- It allows change in the requirements as the project evolves.
- The main focus is quality of the software product.

Drawbacks of RUP

- It fails to provide any clear implementation guidelines.
- RUP leaves the tailoring entirely to the user.

Nine Workflows

There are *6 core* and *3 supporting* process workflows involved in the 4 phases of RUP, which represent a partitioning of all workers and activities into logical groups. You can see these total 9 workflows. We will study about it in this section.

1. **Business Modelling**
   - It is about modelling the ***business context*** and the ***scope*** of your system using use cases. This workflow is part of ***Inception*** and ***Elaboration*** phase.
   - **Active workers during this process :** Business process analyst, Business Designer and Business Model Reviewer.
2. **Understanding Requirements**
   - The purpose of this workflow is to elicit the requirements for the project, including their identification, modeling, and documentation.
   - The goal of this process is the Software Requirements Specification (SRS) which encompasses the captured requirements. This describes what the system should do and allows the developers and the customer to agree on that description.
   - Active **workers during this process :** System analyst and software architect.

   Works of system Analyst :
   - Elicit stakeholders requests
   - Develops requirements management plan
   - Finds actors and use-cases

   Works of software Architect :
   - Prioritize use-cases
   - Review requirements
3. **Analysis and Design**
   - The purpose of this workflow is to ***develop a robust architecture*** for the system based on the requirements, to transform the requirements into a design, and to highlight the major risks (if any).
   - The goal of Analysis is to determine the risks, the stability of the product and expenses of resources.
   - The goal of Design is to show *how* the system will be understood in the implementation phase.
   - Active workers during this process : Architect and Designer

   Works of Architect : Architectural analysis

   Works of Designer :

   - Use-case analysis
   - Use-case design
   - Class design
4. **Implementation**
   - The purpose of implementation is to ***code and test*** the system.
   - The goal of implementation is to develop ready to execute module independent of other modules.
   - **Active workers during this process :** Implementer, integrator and code reviewer.

   Works of Implementer :
   - Implement a module
   - Fix a defect
   - Perform unit testing

   Works of Integrator :
   - Plan system integration
   - Plan subsystem integration
   - Integrate subsystem
   - Integrate system

   Works of Code Reviewer :
   - Review code
5. **Testing**
   - The purpose of the test workflow is to design test case procedures and other verification methods.
   - The goal of test process is to design and execute test cases for the system inorder to eliminate defects.
   - Active workers during this process : Test designer, tester

   Works of Test Designer :
   - Plan test
   - Design test
   - Implement test
   - Evaluate test

   Works of Tester :
   - Execute test
6. **Deployment**
   - The purpose of deploy is to install the software at the end-user.
   - The goal of deployment is to deliver the system to the user community. It involves issues of marketing, packaging, installing, configuring, supporting the user-community, making corrections, etc.
   - **Active workers during this process:** Deployment manager and technical writer.

   Works of Deployment Manager :
   - Develop deployment plan
   - Manage acceptance test
   - Define bill of materials

Works of Technical Writer :

- Write release notes
- Develop support materials

7. **Configuration and Change Management**
   - The purpose of Configuration and Change Management (CM) is to monitor and administrate changes in the project work so that they are consistent with the requirement.
   - The goal of CM is to release and control the version (changes made) of the system.
   - **Active workers during this process:** Configuration Manager and Change Manager.

Works of Configuration Manager :

- Set up CM environment
- Establish CM policies
- Write CM plan

Works of Change Manager :

- Establish change control process
- Review change request

8. **Project management**
   - The purpose of software project management is balancing the competing objectives, managing risk, and overcoming constraints so as to deliver a successful product which meets the needs of both customers and users.
   - **Active workers during this process:** Project manager and Project Reviewer.

Works of Project Manager

- Initiate project
- Develop iteration plan
- Develop quality assurance plan
- Monitor project status
- Schedule and assign work
- Report status
- Handle exceptions and problems

Works of Project Reviewer :

- Project approval review
- Project planning review
- Iteration plan review
- Iteration evaluation criteria review

9. **Environment**
   - This provides the software development environment –both processes and tools - that are needed to support the development team.
   - **Active workers during this process:** Process Engineer, Software architect and Tool Specialist.

Works of Process Engineer :

- Development case
- Project specific templates

Works of Software Architect :

- Design guidelines
- Programming guidelines

Works of Tool Specialist :

- Tool guidelines
- Tools

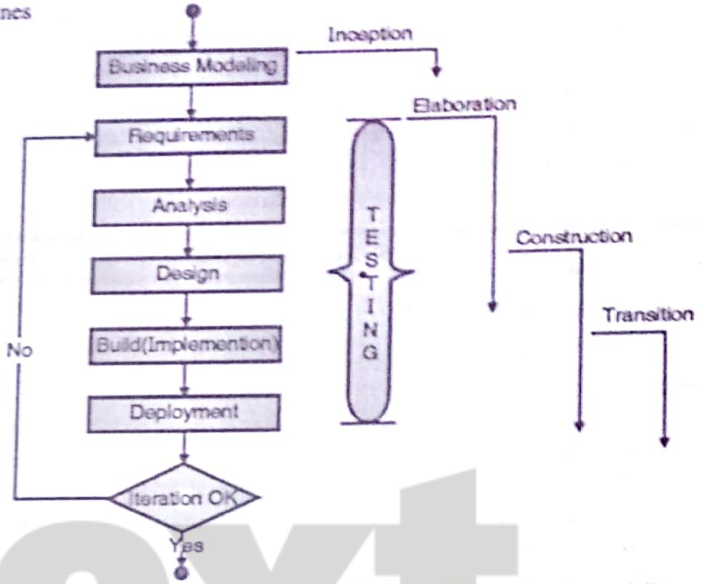


Fig. 2.9.2 : The 4 phases of RUP and their 6 core workflows

Syllabus Topic : Agile Development - Agility

2.10 Agile Development

2.10.1 Agility

Q. What is agility ?

Agility is the unending process, which always accepts specific requirement of the product from the customers or end user then it checks whether the requirements are growth oriented or not ? If it is growth oriented then it aggressively changes the existing system.

The Agile alliance [AG103] defines 12 principles to achieve agility

1. Customer satisfaction by continuous delivery of the software.
2. Always accept changes in requirement though it is late in development.
3. Deliver working software frequently in shorter time span.
4. Business people and developers must work together during complete development of the project.
5. Develop the project from the software project team. It includes;
   - Motivate each individual of the project.
   - Provide then necessary resources and environment to build the project.
   - Trust them to get the job done.
6. Do the face to face conversation, if essential or urgent.
7. Primary measure of the progress is the working software.

- Incremental development is supported through small frequent releases of the system. Where requirements of these increments depend on customer stories and process planning.
- New build of the software is accepted only if all tests execute successfully.
- Change in a system is supported through development, through test first development, as well as through continuous integration.
- XP follows basic framework activities which consist of planning, designing, coding and testing.
- Each activity consists of set of rules and regulation.
- XP process is illustrated in Fig. 2.10.1.

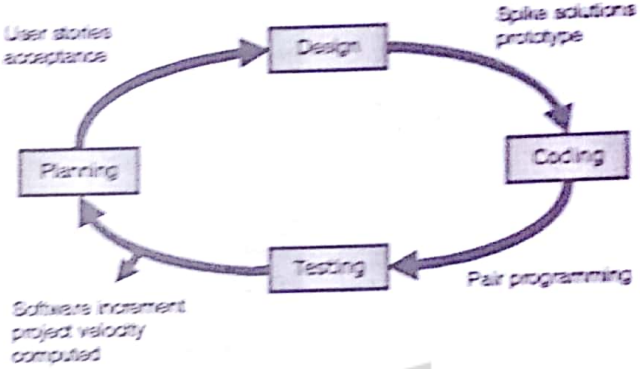


Fig. 2.10.1 : Basic XP process model

- Customer or representative of customer (i.e. may be end user) is involved in through out development process.
  - Customer or representative of customer and end users are the part of agile team so they should be active during the development process.
  - They should be active in defining all requirements of the project.
  - They should be active during the planning also.
  - They may passive during designing and coding.
  - But they are again active during testing if they are accepting the test results then project is to be implemented in actual office site for implementation and maintenance purpose.
- Coding supports pair programming which includes;
  - Collective ownership of the programs
  - Sustainable development of the program
  - It does not involve excessively long working hours.
- Maintaining Simplicity through
  - Simple designing that do not anticipate future enhancement.
  - Constant refactoring to improved quality of code.

Table 2.10.1 : Experts view regarding extreme programming

Experts View
"Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback and courage." **Ron Jeffries**

- The detailed phase of XP process model is

1. Agile process promotes sustainable development. (That means somebody from agile team doing a good task is helpful for development should get promotions).
2. Keeps Continues track to technical excellence and best design increases agility.
3. It collects information about amount of work not done and tries to find out reasons of that.
4. Agile team collects requirement, and fulfil them using best architectures.
5. Continuous thinking about how the performance of team can be increased.

Agility can be applied to any software process.

Syllabus Topic : Agile Process

2.10.2 Agile Process

An Agile software process is characterized in a manner that addresses 3 key assumptions of software project.

1. It is difficult to predict which requirement is changing and which is not changing?
2. Some type of software, design and construction phase are related with each other, so again it is difficult to predict exactly how much time required to design process.
3. Time required for analysis, design and construction are not predictable.

This unpredictability can be managed by;

- Agile software development adapts incrementally.
- It accepts feedback from the customer.
- If requires then use prototype approach.

Agile Process models

- The main objective of agile process model it to provide best quality software in allotted time span.
- It follows conventional approach.
- It follows philosophy and guidelines suggested in manifesto for agile software development.
- There are many Agile Process models that are :
  - XP
  - ASD
  - DSDM
  - Scrum
  - Crystal
  - FDD
  - AM.

Syllabus Topic : Extreme Programming

2.10.3 Extreme Programming

Q. Explain XP in detail.

Extreme programming is most widely used Agile development model; it is also called as XP. It was popular in late 1980's. Idea of this extended programming is developed by the Kent Beck, Jeffries, Beck and Fowler. Extended programming is associated with ideas and methods.

- Extended programming (XP) uses all techniques of object oriented paradigm during the development process.
- XP follows an extreme approach to iterative development.
- New versions of software may be built several times per day and increments are delivered to customers roughly every two weeks.

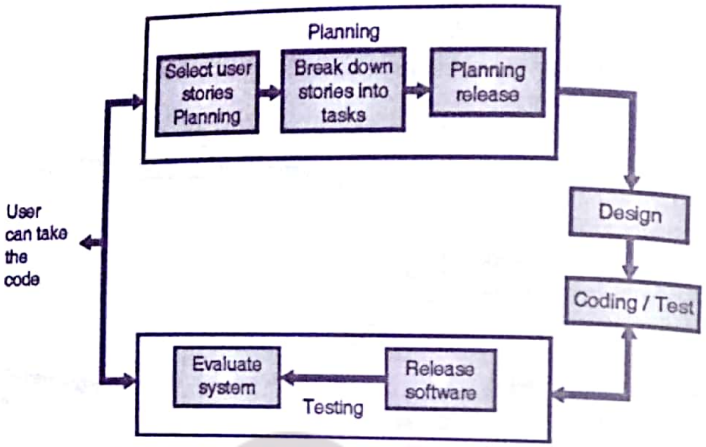


Fig. 2.10.2 : Detailed view of XP process model

I. Planning

Planning activity means gathering the information from customer and plans the project. It consists of :

- Planning activity is begins with creation of set of stories, which consist of features and functionality of the software to be built.
- Each story is written by the customer, which is placed on an index card or story card. Where each index card having;
  - A value assigned by customer which is nothing but priority.
  - Stories (i.e. features and functions) based on business.
  - The value of story may also depend on presence of another story.
  - Example of story card is given below :

Table 2.10.2 : Story card or index card

Downloading and printing an article
First you select article that you want from displayed list. You then have to tell the system how you will pay for it-this can be either from subscription, through company account or by credit card. After this, you get copy write from the system to fill in. When you have submitted this, the article you want is downloaded on to your computer. You choose the printer and a copy of article is printed. You tell the system printing has been successful. If the article is print only article, you cant keep PDF version, so it is automatically deleted from your computer.
Cost of Story : 1 Week

- Other members of XP team assess the each story and assigns cost to each story card. Where,
  - Cost of each story card is nothing but total time span required to complete development of all tasks (i.e. functions and features) mentioned in the card.

  - This time span is measured in hours, days, or weeks.
- If story requires more than three development weeks then customer is asked to split story in to smaller stories and again priorities are assigned and costs are evaluated.
- New stories can be written at any time.
- Customers and XP team work together and to group stories to next increments of the software developed by the XP team.
- Once basic commitment or agreement (It includes delivery date with another project matter) is made up for a release, the XP team orders the stories that will be developed using three ways;
  1. All stories will be developed and implemented quickly using first in first out method.
  2. The story with highest value will be moved up in the schedule and implemented first.
  3. The riskiest stories will be moved up in the schedule and implemented first.
- First increment is called as a first project implementation. Similarly second iteration development is called as a second increment or second project implementation.
- After the delivery of first increment of the project XP team computes project velocity. Where project velocity can be find out as number of stories implemented during the first or one increment in some time span i.e. some few weeks.
- Project velocity is used for
  - Computing the delivery dates.
  - It finds whether over commitment is made or not? And if over commitment is occurs then immediately it again compute the exact delivery dates.
  - As development work proceeds, customer can add new stories.
  - Change the priorities of existing stories.
  - Split stories.
  - Delete stories.
- XP teams decides schedule of development according to project velocity and project priorities.

II. Design

XP design rigorously follows the KIS principles.

- KIS stands for Keep It Simple.
- Design gives guidelines for story writing i.e. nothing less, nothing more..
- The design of extra functioning is discouraged.
- These design guidelines should be followed in every software engineering method although there are times when sophisticated design notation and terminology may get in the way of simplicity.
- XP encourages the CRC (Class Representative Collaboration) cards.
- CRC identifies and organizes the object oriented classes that are relevant to the current software increment. That means it uses inheritance property of object oriented programming language.
- CRC card is nothing but to design work product produced.
- XP design phase converts each story card into CRC card.
- During the design of some story card it founds some difficulties. It follows operational prototyping approach to design complex part of that story card. This is called as a **spike solution.**
  - Spike solution is used to design complex part of the story card.
  - It uses operational prototype to design.
  - It is immediately implemented and evaluated from the end user.
  - So it reduces risk when true implementation is started.
  - It easily validates the original estimate.

- It encourages for refactoring.
  - Refactoring means constructing such a technique which is also used to design techniques.
  - It improves internal structure of the code.
  - It's disciplined way to clean up code.
  - This keeps code simple and maintainable.
  - Easy way to develop new code.
  - It minimizes bugs of the design as well as chance of introducing new bugs in the design.
  - Time required to create new design is very less.
  - Refactoring can be presented before, as well as after coding, that means continuous programming.

Table 2.10.3

Experts View

"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure. It is disciplined way to clean up code [and modify/simplicity the internal design] that minimizes the chances of introducing bugs. In essence, when you refactor you are improving the design of the code after it has written."

Fowler

- Some times, work product is other than CRC card or spike solution that means they can be developed using procedure oriented programming language.
- A central notion of XP is that - design occurs both before and after coding.
- Construction activity provides XP teams guidance on how to improve the design.

III. Coding

Once stories are converted into CRC cards, then XP team moves towards code development. It develops the code using following steps.

- The team exercises on each story and creates a series of unit test.
  - This approach is similar to knowing the previous exam questions before we begin the study.
  - It makes studying easier by focusing attention only on asked questions. Developer is better able to focus on what must be implemented to pass the unit test.
  - In this way, developer can construct a code, which is ready for unit test.
- Once code is complete, it can be unit tested immediately;
  - Test is taken by developer itself;
    1. It checks validity of the all input data.
    2. Whether code is producing expected result or not?
    3. Calls customer or representative of customer or end user to test the output of constructed code.
  - Feedback is collected.
- The key concept of coding activity is pair programming.

Pair programming

XP recommends that two people work together at one computer workstation to create code for a story. It is very innovative practice to distribute story card among the pairs of programmer. They

actually sit together at same workstation to develop the software. Development does not always involve the same pair of people working together. This supports:

- Pairs are created in sucha away that every team member gets a chance to work with every other team member i.e. all team members may work with each other during the development process.
- Mechanism for real time problem solving.
- Mechanism for real time quality assurance.
- Keeps developers focused on the problem at hand.
- Each person of the team plays different role in development.
- One person may think about coding details of particular portion of the code.

  Other person ensures that coding standards are being followed and code that is generated will "fit" into the border design for the story.
- As pair programmers complete there work, the code they developed is integrated with the work of others.
- Sometimes, this is performed on daily basis by the integration team.
- Daily integration gives following advantages:
  - Avoids compatibility.
  - Avoids interfacing problems.
  - Provides smoke testing environment.
  - It early uncovers the errors.

Advantages of pair programming are :

It supports idea of common owner ship and responsibility for the system. It encourages;

- Egoless programming (proposed by Weinberg).
- Software is owned by the team.
- Individual is not held responsible for the problem.
- So individuals do not face that much stress.
- Collectively team has responsible to resolve these problems.

It acts as a informal review process because each line of the code is looked at by at least two people.

- Code inspections and review are very successful.
- Discovering high percentage of software errors.
- It is much cheaper inspection process than formal program inspection.
- Formal program inspections are time consuming but it is not much time consuming.

It helps supports refactoring, which is process of software improvements.

Drawbacks of pair programming

- Each story needs two programmers instead of one.
- If they are not comfortable with each other because of some human factors they will not be develop good software.

IV. Testing

Till now we know that there are two development strategy, Plan based development and Iterative development. Each development uses different testing techniques. Our Agile view process supports iterative development. XP spares more emphasis on than other Agile methods on the testing process.

- Testing procedure is central to XP where approach has been developed that reduces the same functions development in to the existing software.
- The XP Team assesses each story and breaks it into many tasks.

- Each task represents discrete features of the system.
- Unit test for each task are designed.
- Each task can generate one or more unit tests.
- Each unit test describes the test cases.
- The role of customer in the testing process is to help in developing acceptance tests for the st that are to be implemented in the next release of the system.
- Acceptance testing is the process where the system is tested using customer data to check th meets the customer's real needs.
- XP supports incremental testing.

Acceptance test of each story involves

- Making several document selections.
- Paying for them in different ways.
- Printing them on different printers.
- Data validity test.
- Expect output tests.
- Series of different tests rather than single tests.
- Writing code for testing.

The Key advantages of testing in XP are:

1. Test Driven Development (TDD)
2. Incremental test development from scenarios.
3. User involvement in test development and validation.
4. The use of automated tests.

Q. What is Test First Development ?

Test Driven Development (TDD also known as Test First Development) : It is the mo important innovations in XP. Writing tests first implicitly defines both interfaces and a specification b behavior for the functionality being developed. It consists of :

- Test first development is the technique, where test code is written first, and then code is generate for that test.
- This test code includes :
  - Once software is developed, then immediately test can be executed.
  - Testing components should be stand alone.
  - It should simulate submission of the input to be tested.
  - It should check that the result meets the output specification.
- It consists of quick and easy executed tests. This helps to;
  - Checks all functionality of newly added codes.
  - Finds almost all problems of newly added codes.
- Because of test, first development task developers have thoroughly understand the specification of story of customers.
  - Code does not have any ambiguity.
  - Code does not omit any important specification.
- It avoids **'test-lag'** where, because the developer of the system works at faster pace than the tester. That means developer only gives his or her attention to develop and implement the code rather than testing which helps to extreme programming.

Disadvantages of the Test First Code are :

1. Programmers always prefer to write a code for tasks not for test.
2. Some time test codes are incomplete.
3. Some test codes are not checks any exceptional cases.
4. Some tests are very difficult to write because of complexity of the code.
5. Difficult to judge completeness of the test code.
6. Crucial part of the system may not be executed so it remains untested.
7. Customer does not give sufficient time for development.

Incremental test development from scenario :

This is common approach, which is adapted in any development. When coding is over, then code is given to another team for the testing purpose. It consists of following tests.

- Checking each individual developed component of the software.
- Validity of all input data.
- Expected results with respect to the inputs.
- Checking logic of each module.
- Checking all expected features and functionality of the software are accomplished by the code or not ?

User involvement in test development and validation :

This type of test is totally based on end user of the system. It is virtually impossible for a software developer to check whether the software is really use full to end user and how it is useful ? The main objective of the software development is the customer satisfaction. So it includes :

- End user checks all features and functionality of the software.
- All requirements are accomplished or not ?
- It conducts Alpha test and Beta test.
- The Alpha test is conducted at developers site by end user.
- Beta test is conducted at end user's site.

The use of automated test harnesses :

Here automatic test conducting software is built by the developer. This test software is developed after the code development is over.

- Customer and developer forms pair.
- Tests are prepared from pair.
- Test codes are developed by developer.
- Testing is done by user and developer.

Comparison of XP tests :

Comparison of all is given below :

Sr. No.	TDD	Incremental test development	User involvement in test development and validation	Automated test harness
1.	Tests are designed even before the coding begins.	It uses conventional white box testing	It uses conventional black box testing and acceptance testing.	Tests are designed after coding

Sr. No.	TDD	Incremental test development	User involvement in test development and validation	Automated test harness
2.	It ensures all requirements are fulfilled	Not all requirements are fulfilled.	It ensures all requirements are fulfilled	It ensures all requirements are fulfilled
3.	quick and fast	Lengthy	Lengthy	time taking to develop tests but then it is fast.
4.	Less time spent on testing.	More time spent on testing.	More time spent on testing.	More time spent on testing.

Terminologies used in XP :

Terminologies	Description or definition
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. Developers break these stories into development tasks and make the schedule of development according to priorities of stories.
Small releases	– The minimal useful set of a functionality that provides business value is developed first is called as a small release. – Through new increments it adds some extra features or functionality is also called as a small release.
Simple design	Small design is carried out to meet the current requirements is called as a simple design.
Test first development	It is an automated unit test frame work, which is developed before actual task development.
Refactoring	Refactoring is nothing but improving the quality of code continuously such that it can be used in future development.
Pair programming	Each group is developed code one story. Where each group consists of two developers and they are checking each others work.
Collective ownership	Different pair of developers are work on distinct areas of one project. So development ownership of the project is not goes to any single developer. It goes to all developer working on this project so it is called as collective ownership.
Continuous integration	As soon as a pair of developer completes their task with proper testing, immediately that code is integrated into the whole system. After integration, system should pass through the unit test.
Sustainable pace	Large amount of overtime is not acceptable because it reduces quality of the code and medium term productivity.
On-site customer	When customer or representative of the customer or end user is working with XP team for the development as team member of Agile team then this customer is called as on-site customer.

Application of XP

- Planning game
- Small releases
- Customer acceptance tests
- Simple design
- Pair-driving programming
- Test driving development
- Refactoring the program / code
- Continuous progressive integration
- Collective code ownership
- Use of coding standards
- Metaphor for modeling
- Maintaining sustainable pace of development.

Review Questions

Q. 1 Explain in brief the various types of Software Process.
Q. 2 List the umbrella activities followed in generic process model.
Q. 3 Explain waterfall model in brief.
Q. 4 What is the need of feasibility study? Explain its types.
Q. 5 List the advantages and drawbacks of Incremental model
Q. 6 State the difference between :
(a) Classic life cycle model and Prototyping model
(b) Waterfall model and Spiral model
Q. 7 What does the radius imply in Spiral model ?
Q. 8 State the purpose, advantages and drawbacks of RUP?
Q. 9 What is agility ? Explain XP in detail.
Q. 10 What is Test First Development ?

❑❑❑

UNIT I

CHAPTER 3

Requirement Analysis and System Modeling

Syllabus :

Requirements Engineering, Eliciting Requirements, SRS Validation, Components of SRS, Characteristics of SRS.

3.1 Requirements

- A *Requirement* is a condition needed by a user to solve a problem or achieve an objective. For *example*, a requirement for a car could be that the maximum speed to be at least 120 mph.

System requirements fall into two categories

1. Functional 2. Nonfunctional

Table 3.1.1 : Difference between functional and nonfunctional requirements

Sr. No.	Functional Requirements	Nonfunctional Requirements
1.	Describes Product behavior	Describes Product's quality attributes.
2.	Describes what the product should do i.e. the actions/work/services of the software.	Describes capabilities of the product i.e. how the software should behave to meet the user needs.
3.	Describe the services that a system should provide	Describes the design and implementation constraints on the services and the external interface which a product must have.
4.	Describe 'what' the system should do in a particular condition.	Describe 'how' the system should work so as to be user friendly and secured.
5	Example : A bank software has functions – open acc., close acc, withdraw, loan claim etc.	Example : Performance, reliability, portability, security – system must run on windows server 2003, system must be secured against Trojan attacks.
6	The functions are represented using use case diagram & use case specifications.	The performance, usability, reliability, and security quality attributes are documented in narrative descriptions.

Syllabus Topic : Requirements Engineering

3.2 Requirements Engineering

Requirement Engineering is a bridge to design and construction of a quality software product.

- Requirement Engineering establishes an interaction between the developers, customers and users.
- The *input* to requirement engineering is : wishes, problems, unclear requirements etc. and the *output* of requirement engineering is the *Requirement Specification* and *Analysis model* that includes complete coverage of the problem and complete-exact definition of each requirement.
- Requirement Engineering is collecting the information (requirements) about what the system should do (not how to do it).

Requirement engineering includes following tasks so as to improve Software quality :

Task 1 : Inception : Defines the scope and nature of the problem to be solved.

Task 2 : Elicitation : Helps the customer to define what is required.

Task 3 : Elaboration : Customer's basic requirements are refined and modified.

Task 4 : Negotiation : As the customer defines the problems, negotiation occurs by highlighting the priorities, what is essential and what isn't, when it is required and many such.

Task 5 : Specification (SRS): Finally, the proposed requirements are specified. Every project needs requirements specification document because it is the formal agreement between the client/end-users, the business owner/stakeholder and the project manager. It states exactly what should and should not be included in a project and what the end-user expects from the proposed project.

Task 6 : Validation : Ensures that both customer's and engineer's understanding of the problem coincide.

Task 7 : Management : Helps the software engineers to control and track the changes in requirements at any time as the project proceeds.

- The above Requirements engineering tasks are important in software development projects as it influences the development cost, time, effort and quality.
- SRS includes the demands of stakeholders (customers, managers or end users). They specify the desired functions, quality attributes and other properties of the proposed software that is to be built or assembled.
- SRS helps software engineers to better understand the problem they will work to solve. SRS involves analyzing and accurately representing the client's requirements in a manner that can be effectively implemented in a system that will confirm to the client's specifications.
- But, in many situations, enough care is not taken in establishing correct requirements. This causes problems, later, in the development life cycle, and more time and money is spent in fixing these problems. Thus, it is necessary that requirements are established in a systematic way to ensure their accuracy and completeness. Requirement Analysts require both communication and technical skills.

Example : A Library Management System is to be designed so that information on books, CDs, DVDs, Journals, etc. can be stored and retrieved.

Requirements demanded by the client i.e. 'what system should do' are:

1. *Functional requirement*: Searching by Title, Author and ISDN should be possible.
2. *Implementation requirement*: User Interface should be web based i.e. accessible via web browser.

3. *Performance requirement*: At least 20 transactions per second should be possible.
4. *Security requirement*: Users shouldn't have access to personal data of other users.

There are few requirements that are not told by the client but decided by Software Engineer :

1. System structure.
2. Implementation technology or language.
3. Development environment or methodology.

Deliverables (Output) of Requirement Gathering tasks

The requirement gathering task helps in obtaining the following results :

- Identifies Entities.
- Essential Use Cases.
- Possibly workflow diagrams, flow charts.
- ER Model – Data Objects (entities), properties of objects (attributes), operations on objects and relationships between objects.
- We are going to study 'how to generate all these outputs' in up-coming chapters.

3.3 Requirements Engineering Tasks

Requirement Engineering is a process used to discover what the customer needs, analyse those needs, assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as the project proceeds. It is accomplished through the execution of 7 distinct tasks mentioned in the Section 3.2 and are detailed in the following sub sections.

3.3.1 Inception

- A new project is started when a new business need is identified or a new service is discovered
- The stakeholders of the system perform feasibility study to decide whether or not the proposed system is useful. The study checks :
  - Will the system add to organisational benefits ?
  - Can the system be engineered using current technology and within budget ?
  - Can the system be integrated with other systems that are used ?
- At project Inception time, software engineers ask some questions for people in the organisation based on information collected :
  - What if the system wasn't implemented ?
  - What are current process problems ?
  - How will the proposed system help ?
  - What will be the integration problems ?
  - Is new technology needed ?
  - What facilities must be supported by the proposed system ?
- The need of these questions is to find :
  - The effectiveness of collaboration between the customer and developer.
  - Basic understanding of the problem.
  - Who will use the solution ?
  - The desired nature of the solution.

Syllabus Topic : Eliciting Requirements

3.3.2 Eliciting Requirements

- Requirements Elicitation is also called as Requirements Discovery. It involves identifying the application domain and the services that the system should provide.
- Problems of requirements gathering (eliciting) :
  - Problems of Understanding :
    - End users many a times don't know what they really want
    - End users express requirements in their own terms
    - Different stakeholders may have different requirements
  - Problems of Scope :
    - The scope of the system is not defined properly as the customers specify unattainable requirements that may confuse, rather than clarify, the overall system's objectives.
    - Organisational and political factors may influence the system requirements.
  - Problems of Volatility :

    The requirements change during the analysis process. New stakeholders may emerge and the business environment changes.

To avoid these problems, requirements engineers must approach the requirement gathering activity in an organized manner.

3.3.3 Elaboration

- Elaboration is about creating an analysis model that defines the informational, functional and behavioral aspects of the problem.
- The main task is to describe the problem in a way that establishes a firm base for designing a model.
- Elaboration focuses on expanding the information (i.e. obtained from inception and elicitation) and then developing a refined technical model of software functions, features and constraints (i.e. restrictions or limitations). This process is composed of various modeling and refinement tasks.
- Different models may be produced during this activity depending on the relationships and collaboration between the various business domain entities.
- From the designed model, it would be easy to judge if the efficiency of workflow of the system is as it has been imagined.

3.3.4 Negotiation

- It happens that different stakeholders (customers, business managers or users) propose conflicting requirements. In such a situation, the requirements engineer must settle these conflicts through a process of negotiation.
- First, the key stakeholders are identified. These are the people who are will be involved in negotiation. Then, the customers, users and other stakeholders are asked to rank the requirements and then discuss about the requirements according to their priority in order to settle out the conflicts between them. Risks associated with each requirement are identified, analyzed and

accordingly modified and alternatives are developed so that each party gets satisfied. So, negotiation works towards a set of requirements that makes all the parties to win.

- Negotiation process involves - Requirements collection (from different stakeholders), Classification, Prioritisation, Conflict resolution, Requirements checking and Finalizing the requirements.
- Finally, rough estimates of development effort are made to calculate the project cost and delivery time.

3.3.5 Software Requirement Specification (SRS)

- Specification is concerned with documenting the requirements and this document is maintained over the life of the project. This is the most fundamental condition for successful implementation of the project.
- Specification is not limited to just text document, it can include graphical notations, mathematical specifications, user scenarios or prototypes.
- Specification includes a set of use cases that describe how the user interacts with the software. Use cases are also known as functional requirements. In addition to use cases, specification also includes nonfunctional requirements such as performance requirements and quality standards. (Example of requirements is given in Section 3.2).
- Specification is the complete description of behavior of the system to be developed. It is the final deliverable of requirements engineering task which serves as an artifact for further software engineering activities.

3.3.6 Requirements Validation

- Validation is concerned with checking if the specification meets the users' needs.
- Software engineer checks the specification to confirm whether he is building the right product ?
- As each element of the analysis model is created, it is validated for consistency, omissions and ambiguity. A review of the analysis model addresses the following questions :
  - Is each requirement consistent ?
  - Are all requirements specified clearly ?
  - Is the requirement really necessary and also is it attainable ?
  - Is each requirement bounded and unambiguous ?
  - Can each requirement be traced with its source ?
  - Is each requirement testable ?
  - Are the requirements very complex and if so is it broken into simple and understandable modules ?
- Have all requirement patterns been properly validated against the customer requirements ?
- Validation mechanism is the formal technical review that includes software engineers and different stakeholders who examine the specification to ensure that all the requirements have been stated clearly, errors are detected and removed, looks if there is any missing information, any inconsistencies, conflicting requirements or unrealistic requirements.
- Validation is concerned with checks for comprehensibility, validity, consistency, completeness, realism, verifiability, necessity, traceability and adaptability.

Thus, the requirement validation checklist is as follows :

- **Comprehensibility :** Are requirements stated clearly and properly understood by all stakeholders ?

- *Correct* - A set of requirements is correct if and only if every requirement stated therein represents something required of the system to be built." Davis (1993)
- *Unambiguous* - The reader of a requirement statement should be able to draw only one interpretation of it.
- **Incorrect Example** – "The system should not accept password longer than 8 characters"

  The above stated requirement can have multiple interpretation as given below :
  - The system should not allow user to enter more than 8 characters in the password field
  - The system should truncate the entered data to 8 characters
  - The system should display an error message if user enters more than 8 characters in the password field
- **Correct Example** – "The system should not accept passwords longer than 8 characters in the password field. If the user enters more than 8 characters while entering the password, an error message should prompt the user to correct the same."
- *Complete* - No requirements should be missing. A complete requirement leaves no room for guessing. Include all significant requirements, whether related to functionality, performance, design constraints, attributes, or external interfaces
- *Consistent* - A consistent requirement does not conflict with other requirements in the requirement specification.

  Incorrect Example :

  REQ1 – "The birth date should be entered in mm/dd/yyyy format"

  REQ2 - "The birth date should be entered in dd/mm/yyyy format"

  Correct Example

  REQ1 – "For the US users the birth date should be entered in mm/dd/yyyy format"

  REQ2 – "For the French users the birth date should be entered in mm/dd/yyyy format"
- *Ranked for Importance* – Requirements should be achievable. But sometimes, few requirements are not attainable.

3.3.7 Requirements Management

- Requirements management is a mechanism to control and track the changing requirements at any time as the project proceeds.
- The requirements change or say, new requirements emerge during the process and this may be because;
  - The business needs change and a better understanding of the system is developed.
  - The priority of requirements from different viewpoints changes during the development process.
- Each requirement is assigned a unique identifier i.e. useful in in developing the *traceability tables*. Each requirement traces to one or more aspects of the system.. This is shown in the Table 3.3.1;

Table 3.3.1 : A Traceability table

Requirements	Specific Aspects of the System					
	A1	A2	A3	.	.	Al
R1	✔		✔			
R2			✔	✔		✔
.		✔				
.	✔				✔	
Rn		✔	✔			

- Few traceability tables are as below :
  - **Features traceability table :** Shows how requirements relate to important product features.
  - **Source traceability table :** Identifies the requirements and the source (stakeholders) who proposed these requirements.
  - **Dependency traceability table :** Indicates how requirements are related or depended on one another.
  - **Subsystem traceability table :** Categorizes requirements by the subsystems that they govern.
  - **Interface traceability table :** Shows how requirements relate to both internal and external system interfaces.
- Traceability is useful in case of large and complex system to determine the connections between the requirements so as to understand how a change in one requirement affects the other requirement and how it will affect the different aspects of the system to be built.

3.4 Requirement Elicitation

- It is about collecting the requirements, needs and constraints.
- This task collects the information about :
  - Domain - problems and laws
  - Existing standards and systems
  - Existing specifications
- The Q &A session discussed in Section 3.3.1 (or during inception) does not lead to a very detail elicitation of requirements and so the following approaches are used for successfully eliciting the requirements.

3.4.1 Collaborative Requirements Gathering

- The Q & A session during inception only establishes the scope of the problem and overall perception of a solution. Out of these initial meetings, the stakeholders write a one or two page "product request". A meeting place, time, and date are selected and a facilitator is chosen. The software team and the other stakeholders are invited to attend this meeting. The product request is

distributed to all attendees before the meeting date. Before coming to the meeting, each attendee is asked to make a list of objects that are part of environment that surrounds the system, that are to be produced by the system and objects that are used by the system to perform its functions. Also, he is asked to list the services or functions that act on these objects. Finally, list of constraints (like business rules, cost and size) and performance criteria (i.e. speed, accuracy) are also developed.

- Then comes the actual meeting (collaborative requirement gathering) date. In this gathering, a team of stakeholders and developers work together to identify the problem, propose elements of solution, negotiate different approaches and specify a preliminary set of solution requirements.
- A collaborative requirement gathering is about :
  - Conducting meetings which are attended by both software engineers and customers.
  - Rules for preparation and participation are established.
  - An agenda is suggested to cover all important points and encourage the free flow of ideas.
  - A 'facilitator' (may be a customer or developer) controls the meeting.
  - A 'definition mechanism'(can be worksheets, flip charts, wall stickers or an electronic bulletin board, chat room or virtual forum) is used.
- The goal of such gathering is :
  - To identify the problem
  - Propose elements of the solution
  - Negotiate different approaches Specify a preliminary set of solution requirements.

Example : Consider the "Safe Home" project.

- The list of *objects* involved in this project are control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event (sensor has been activated), a display, a PC, telephone numbers, a telephone call and so on.
- The list of *services* might include configuring the system, setting the alarm, monitoring the sensors, dialing the phone, programming the control panel and reading the display. All these services act on objects.
- The list of *constraints* might be as - the system must recognize when sensors are not operating, must be user-friendly, must interface directly to a standard phone line.
- The *performance criteria* might be as - a sensor event should be recognized within a second, an event priority scheme should be implemented.

As the *gathering begins*, first topic of discussion is the need and justification for the new product - everyone should agree that the product is justified. Once the agreement has been established, each participant presents his list for discussion. The lists can be pinned to the walls of the room and they can also be posted on an electronic bulletin board or in a chat room environment. Each listed entry should be capable of being manipulated separately (deleted, added or modified). At this stage, critique and debate are strictly prohibited.

Then, a combined list is created by the group so as to eliminate the redundant entries and adds any new ideas that come up during the discussion. Next, the facilitator coordinates the discussion to shorten, lengthen or reword the combined list so as to properly reflect the system to be developed. Now, the team is divided into smaller sub teams; each works to develop mini specifications for one or more entries of the list. Each mini specification is the elaboration of the word included in the list. For example, mini specification of the control panel is - It is a wall mounted unit and is 9*5 inches in size, has wireless connectivity to sensors and a PC, user interaction occurs through a keyboard having 12 keys, a 2*2 inch CD display provides the user feedback, provides interactive prompts, echo and similar functions.

After the mini-specs are completed, each participant makes a list of validation criteria for the system. All these lists are combined and a consensus list of validation criteria is created. Finally, one or more participants are assigned the task of writing a complete draft specification using all inputs from the meeting.

3.4.2 Quality Function Deployment

- It would be very helpful to have a technique that can assist in accurate requirement gathering. Quality Function Deployment (QFD) is a technique that enables a software team to specify clearly the customer's wants and needs, and then evaluate each proposed object or service capability in terms of its impact on meeting those needs.
- QFD is a technique that translates customer requirements into technical requirements for software. QFD focuses on customer needs throughout the software engineering process.

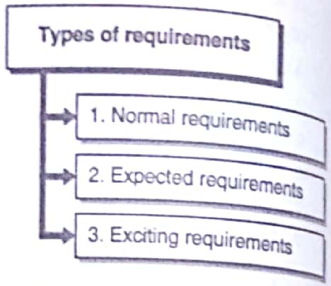


Fig. C3.1 : Types of requirements

QFD identifies 3 types of requirements :

➜ **1. Normal requirements :**

It reflects objectives and goals stated for a system during meetings with the customer. If these requirements are present, the customer is satisfied.

Example : Graphical displays, specific system functions and defined levels of performance.

➜ **2. Expected requirements :**

These are implicit to the system and may be so fundamental that the customer does not explicitly state them. Absence of such requirements causes customer dissatisfaction.

Example : Ease of human-machine interaction, overall operational correctness and reliability and ease of software installation.

➜ **3. Exciting requirements :**

These reflect features that go beyond the customer's expectations and prove to be very satisfying when present.

Example : Customer may request a Word processing software with standard features but he is pleased when he sees that the product delivered to him has number of page layout capabilities along with basic features.

- In meetings with the customer, function deployment determines the value of each function that is required for the system. Information deployment identifies data objects and events that the system must consume and produce. Finally, task deployment examines the behaviour of the system. Then value analysis is conducted to determine the relative priority of requirements determined during each of the three deployments.

QFD uses customer interviews and observation, surveys, and examination of historical data as raw data for the requirements gathering activity. These data are then translated into a table of requirements called the customer voice table. A variety of diagrams, matrices and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements.

Benefits of QFD

- Improves user involvement.

- Improves management support and involvement.
- Shortens the development lifecycle.
- Improves project development.
- Supports team involvement .
- Structures communication processes.
- Provides a preventive tool for improving quality.
- Avoids loss of information.

3.4.3 User Scenarios

- Scenarios are descriptions of how a system is used in practice. Scenarios are real-life examples of how a system can be used.
- Scenarios are particularly useful for adding detail to an outline requirements description.
- The scenario starts with an outline of the interaction, and during elicitation, details are added to create a complete description of that interaction. In general, scenario includes :
  - Description of System state and user's expectations at the beginning of the scenario.
  - The normal flow of events.
  - What can go wrong and how this can be handled.
  - Information about other concurrent activities.
  - Description of System state on completion of the scenario.
- Collection of different user scenarios forms a use case.
- Each scenario is described from the point-of-view of an *actor* - a person or a device that interacts with the software directly or indirectly.
- Each scenario answers the following questions :
  - Who are the primary and secondary actors ?
  - What are the actor's goals ?
  - What preconditions should exist before the user story begins ?
  - What are the main functions performed by the actor ?
  - What exceptions might be considered ?
  - What variations (changes) in the actor's interactions are possible ?
  - What information does the actor desire from the system ?
  - Will the actor have to inform the system about the changes in the external environment ?
  - Does the actor wish to be informed about unexpected changes ?

Example : Safe Home Project

In this example, we consider homeowner as the primary actor; and system administrator and sensors are the secondary actors.

Homeowner interacts with the safe home security function using the control panel.

The basic use-case template for the safe home security system activation is as follows :

Use-case :	Initiate Security Monitoring system
Primary actor :	Homeowner
Goal in context :	To set the security system monitoring sensors when the homeowner leaves the house.

Preconditions :	System's password needs to be set and recognize various sensors.
Trigger :	Homeowner decodes to turn on the alarm functions.
Scenario :	Homeowner observes control panel. Homeowner enters password. Homeowner selects 'stay' or 'away'. Homeowner presses the panic button in an emergency. Homeowner activates the security system.
Exceptions :	Control panel is not ready if any of the sensors are open. Password is incorrect. Stay is selected and perimeter sensors are activated. Away is selected and all sensors are activated.
Interface :	control panel.
Secondary actors :	Support technician, sensors.
Queries / Issues:	Can we activate the system without the use of a password ? Is there any way to deactivate the system before it actually activates?
Interface :	control panel.
Secondary actors :	Support technician, sensors.

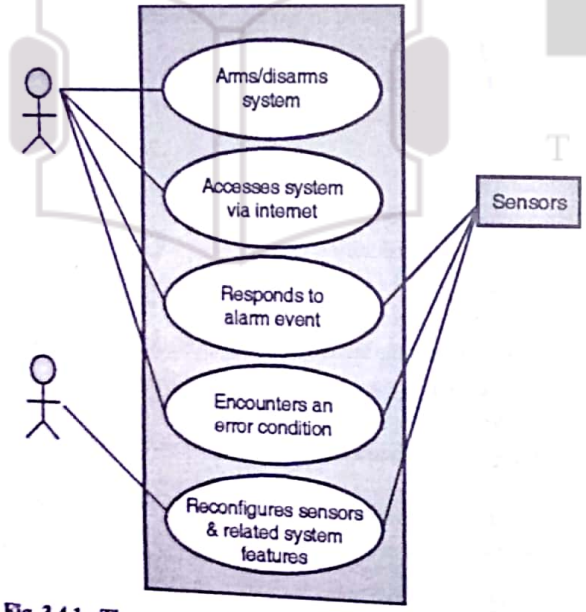


Fig. 3.4.1 : The use-case diagram for safe-home security function

3.4.4 Elicitation Work Products

This is developed as a result of requirements elicitation and it varies depending on the size of the system to be built. It describes :

- The need and feasibility of the system.
- The scope of the system.
- The list of customers, users and other stakeholders who participated in requirements elicitation.
- The technical environment i.e. required.
- List of requirements (that are organized according to their function) and the domain constraints those apply to each.
- A set of user scenarios that provide information about the usage of the system under different operating conditions.
- Any prototypes developed to better define the requirements.

Each of these work products is reviewed by all the people who have participated in requirements elicitation.

3.5 Software Requirement Specification

- Specification is concerned with documenting the requirements and this document is maintained over the life of the project. This is the most fundamental condition for successful implementation of the project.
- Specification is not limited to just text document, it can include graphical notations, mathematical specifications, user scenarios or prototypes.
- Specification includes a set of use cases that describe how the user interacts with the software. Use cases are also known as functional requirements. In addition to use cases, specification also includes nonfunctional requirements such as performance requirements and quality standards. (Example of requirements is given in Section 3.2).
- Specification is the complete description of the behavior of the system to be developed. It is the final work product produced by the requirements engineer which serves as the foundation for subsequent software engineering activities.

☞ **The basic issues that the SRS shall address are the following :**

(i) **Functionality :** behaviour or services provided by the software

(ii) **External interfaces :** external interactions of the software such as with end users, system's hardware or external software

(iii) **Performance :** the throughput, the availability, the response time, the recovery time of the software.

(iv) **Quality attributes :** portability, correctness, maintainability, security, etc.

(v) **Constraints :** implementation language, policies for database integrity, resource limits, operating environments etc.

Example : The ATM system is dependent on the network in the village areas. Due to flood and consequent network jams there will be a delay in the transactions

3.5.1 Benefits (Need) of SRS

- **SRS forms the basis for agreement between customers and development organization on what the software product is expected to do.** Complete descriptions of the functions that are

expected from the software are specified in the SRS. This will help the end users to verify whether the software meets the specified needs or not and if not, then how the software must be manipulated so as to meet their requirements.

- **SRS reduces the development effort.** The work of preparing SRS forces various stakeholders various concerned groups in the customer's organization to think thoroughly of all the requirements before the project design begins, thus reducing the efforts needed for redesigning recoding, and retesting. Careful inspection of the requirements specified in SRS can reveal the outs, misinterpretations and inconsistencies early in the development cycle; hence it becomes easy to correct these problems.
- **SRS forms a base for cost estimation and project scheduling.** As complete description of the product to be developed is specified in the SRS, it helps in estimating the project costs and can be used to obtain approval from the customer for the decided price estimates.
- **SRS provides a baseline for verification and validation.** Software development teams develop their verification and validation plans or say, test plans much more effectively from a well prepared SRS document. As SRS provides a baseline against which requirement confirmation can be measured.
- **SRS facilitates transfer of new software to new environments.** SRS makes it easier to transfer the software product to new clients or new hardware machines. Therefore, developers feel easier to transfer the software to new clients and customers feel easier to transfer the software on other systems of their organization.
- **SRS serves as a basis for software improvement.** SRS describes the product features and not the process of project development; therefore, it serves as a basis for enhancements by allowing the developers to do any later modification if required on the finished product. But then, SRS needs to be altered in that case i.e. SRS forms a base for continuous product evaluation.

Syllabus Topic : Characteristics of SRS

3.5.2 Characteristics of SRS

Great SRS leads to Great Product: We have to keep in mind that the goal is to create great product and great software. A great product can be created only from a great specification. Systems and software these days are so complex that to get on with the design before knowing what you are going to build is foolish and risky.

A great SRS has following quality factors :

- **Correct :** A set of requirements is correct if and only if every requirement stated therein represents something required of the system to be built." Davis (1993)
- **Unambiguous :** The reader of a requirement statement should be able to draw only one interpretation of it.
- **Incorrect Example** – "The system should not accept password longer than 8 characters"

  The above stated requirement can have multiple interpretation as given below :
- The system should not allow user to enter more than 8 characters in the password field
- The system should truncate the entered data to 8 characters
- The system should display an error message if user enters more than 8 characters in the password field

Correct Example – "The system should not accept passwords longer than 8 characters in the password field. If the user enters more than 8 characters while entering the password, an error message should prompt the user to correct the same."

Complete - No requirements should be missing. A complete requirement leaves no room for guessing. Include all significant requirements, whether related to functionality, performance, design constraints, attributes, or external interfaces

Consistent - A consistent requirement does not conflict with other requirements in the requirement specification.

Incorrect Example :

REQ1 – "The birth date should be entered in mm/dd/yyyy format"

REQ2 - "The birth date should be entered in dd/mm/yyyy format"

Correct Example

REQ1 – "For the US users the birth date should be entered in mm/dd/yyyy format"

REQ2 – "For the French users the birth date should be entered in mm/dd/yyyy format"

- *Ranked for Importance* – Requirements should be achievable. But sometimes, few requirements are not attainable.
- *Verifiable* - Don't put in requirements like – "It should provide the user a fast response" or "The system should never crash". Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."
- *Traceable* - Each requirement should be expressed only once and should not overlap with another requirement. Every requirement has a unique identification number so that requirements can be easily traced through out the specification.
- **Adaptability :** Can the requirement be changed without a large impact on other requirements ?

Syllabus Topic : Components of SRS

3.5.3 Components of SRS

Following is the IEEE standards SRS format which depicts the components of a SRS document.

1. INTRODUCTION
   - 1.1 PRODUCT OVERVIEW
     (Product description).
   - 1.2 PURPOSE
     (Product description)
   - 1.3 SCOPE
     Write the benefits, objectives, and goals.
   - 1.4 AUDIENCE
     (lists the users of SRS such as developers, project managers, marketing staff, users, testers, and documentation writers.).
   - 1.5 DEFINITIONS, ACRONYMS AND ABBREVIATIONS
     (defines all the terms necessary to properly interpret the SRS).

1.6 CONVENTIONS

(describes the standard conventions followed while writing this SRS such as fonts or special significant highlights) .

1.7 REFERENCES

(lists the reference documents or web page address used as reference)

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

(lists the reference documents or web page address used as reference)

2.2 PRODUCT FUNCTIONALITY

(designs such as DFDs that describe major functions of the system).

2.3 USER CHARACTERISTICS

Identify various users who will be using this product. Differentiate them based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.

2.4 OPERATING ENVIRONMENT

(Designs that describe that shows the major components of the overall system, subsystem interconnections, and external interface.).

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

Describe the issues of the developers such as - hardware limitations, interfaces to other applications, specific technologies, tools, and databases to be used, language requirements, communications protocols, security considerations, design conventions.

2.6 USER DOCUMENTATION

List out the user manuals, on-line help, and tutorials that will be delivered along with the software.

2.7 ASSUMPTIONS AND DEPENDENCIES

List any assumed factors that could affect the requirements stated in the SRS. These include third-party components or issues around the development or operating environment. The project could be affected if these assumptions are incorrect, are not shared, or change.

Also list out the dependencies such as software components that you plan to reuse from another project.

3. SPECIFIC REQUIREMENTS

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 USER INTERFACES

List out the different screens that will be available to the user.

3.1.2 HARDWARE INTERFACES

List out any special libraries that are used to communicate with your software mention them.

3.1.3 SOFTWARE INTERFACES

(Describes databases, operating systems and tools, services and the data that will be shared across software components)

3.1.4 COMMUNICATIONS REQUIREMENTS

(Describes the communication related requirements such as e-mail, web browser, network server communications protocols, electronic forms and the communication standards such as FTP or HTTP.)

3.2 FUNCTIONAL REQUIREMENTS

(describes the functional requirements in detail with proper explanations regarding each and every function.)

3.3 BEHAVIOUR REQUIREMENTS

3.3.1 USE CASE VIEW

Draw a use case diagram that will illustrate the entire system and all possible actors.

4. OTHER NON FUNCTIONAL REQUIREMENTS

4.1 RELIABILITY

4.2 AVAILABILITY

4.3 SAFETY and SECURITY

(describes the functional requirements in detail with proper explanations regarding each and every function.)

4.4 MAINTAINABILITY

4.5 PORTABILITY

4.6 PERFORMANCE

Provide at least 5 different performance requirement such as -"Any transaction will not take more than 10 seconds".

5. SUPPORTING INFORMATION

This section is Optional. Define any other information/requirements not stated elsewhere in the SRS. This might include internationalization requirements, legal requirements, reuse objectives for the project, and so on.

3.6 Identifying the Stakeholders

- Stakeholder is the personnel who benefits in a direct or indirect way from the system that is being developed.
- Different stakeholders included in the Requirement Engineering/Analysis process are:

1. Client /Customer

- He is the contract partner who orders the software.
- Decides on budget and system functionalities

2. Users

Uses the system

3. Advocate

- Speaks two languages – one of the customer and one of the engineers.
- Should be technically expert and domain expert.

4. **Project Manager**
   - Is the contract partner.
   - Controls the budget.
5. **Requirement Engineer/ Programmer/ Software engineer**

Has knowledge of software engineering and is actually responsible for software development.

☞ **Other stakeholders will include**

- Maintenance operators : anyone who operates the system
- Financial, Safety and Other regulators : organizations which regulate the aspects of the system
- Those organizations who integrate horizontally with the organization for whom the developer is designing the system.

Example : ATM Stakeholders

- **Bank customers :** receive services from the system.
- **Representatives of other banks :** have reciprocal agreements that allow each other's ATMs to be used.
- **Bank managers :** obtain management information from the system.
- **Counter staff :** are involved in the day-to-day running of the system.
- **Database administrators :** are responsible for integrating the system with the bank's customer database.
- **Security managers :** ensure that the system will not pose a security hazard.
- **Marketing department :** uses the system as a means of marketing the bank.
- **Hardware and software maintenance engineers :** are responsible for maintaining and upgrading the hardware and software.

3.7 Issues of Requirement Gathering

Requirement Gathering for the proposed new software needs to identify :

- The technical staff working with customers (stakeholders) so as to find out about the application domain.
- The services that the system should provide.
- The system's operational constraints i.e. how the system is to be used on a day-to-day basis.
- But this all is very difficult to find out as number of problems are faced while requirement gathering.

☞ **Problems of requirements gathering**

There are also, possible problems caused by engineers and developers during requirement analysis :

☞ **Problems of Understanding**

- Stakeholders don't know what they really require. They do not have a clear idea of their requirements.
- Stakeholders express requirements in their own terms. Thus, there is a conflict of opinions between the customer and the developer about the system to be built.

- Different stakeholders may have different requirements.

- Nearly, 40% to 60% of software failures and defects are the result of poor software management and requirements definition. This means, about half of the problems encountered could have been avoided by making it clear, from the very beginning, what the customer expected from the respective project. This means, the programming was fine and the developers did their job well - only they did a different job from what they were supposed to.

☞ **Problems of Scope**

- The scope of the system is not defined properly as the customers specify unnecessary technical detail that may confuse, rather than clarify, the overall system's objectives.
- Organisational and political factors may influence the system requirements.

☞ **Problems of Volatility**

- The requirements change during the analysis process. New stakeholders may emerge and the business environment changes.
- Users or Customers do not be committed to a set of written requirements. Users insist on new requirements in later SDLC phases even after the cost and schedule has been fixed.

- This leads to the situation where user requirements keep changing even when system or product development has been started.

☞ **Problems of Engineer and Developers**

- Technical person and end users may have different understanding of the proposed project and therefore, they may wrongly believe they are in perfect agreement until the finished product is delivered.
- Engineers and developers may try to fit the requirements in an existing system, rather than developing a system i.e. specific to the needs of the client.
- System Analysis may be often carried out by engineers who don't have personal skills and the domain knowledge of the proposed project.

To avoid these problems, requirements engineers must approach the requirement gathering activity in an organized manner. Solution to avoid these issues is to use techniques like prototyping, Unified Modeling Language (UML), use cases, and Agile software development where customer is an active participant from the beginning to throughout the software development cycle.

3.8 Techniques of Requirement Gathering

There are few well-recognised techniques for gathering the information needed in a SRS.

3.8.1 Questionnaires

Different types of questions are asked to different stakeholders at the time of *questionnaire*. Some of these are discussed below :

- Questions asked by the requirement engineer enables the software team to better understand the problem and allows the customer to put his views about the solution (s/w) :
  - How would you characterize "good" output that would be generated by the solution ?
  - What business operations you do ?
  - What steps do you follow to do it i.e. how do you do it ?

- What problems will the solution deal with ?
- In which business environment, will the solution be used ?
- Will special performance issues or constraints affect the way the solution is approached ?

Example : Consider the "Safe Home" project.

- The list of *objects* described for safe home might include the control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event (sensor has been activated), a display, a PC, telephone numbers, a telephone call and so on.
- The list of *services* might include configuring the system, setting the alarm, monitoring the sensors, dialing the phone, programming the control panel and reading the display. All these services act on objects.
- The list of *constraints* might be as – the system must recognize when sensors are not operating, must be user-friendly, must interface directly to a standard phone line.
- The *performance criteria* might be as - a sensor event should be recognized within a second, an event priority scheme should be implemented.
- Another set of questions focus on the effectiveness of communication activity. This set of questions is also called as "*meta questions*":
  - Are you the right person to answer these questions? Are your answers 'official'?
  - Am I asking you the relevant questions to the problem that you have ?
  - Am I asking too many questions ?
  - Can anyone else provide additional information ?
  - Should I ask you anything else ?

The need of these questions is to '*break the ice*' and initiate the communication between the stakeholders and the requirement engineer that is essential for successful elicitation.

3.8.2 Reviews

Reviewing the existing documents and learning about the existing systems that match the proposed system is necessary to get extra detailed information. This will help in understanding the proposed project better.

Reviews can be obtained by following task:

- Analysts should dialog with users of existing systems that are similar to proposed system.
- Analysts must read documentation on existing system
- Review the reports, forms, procedures, descriptions of other companies.
- Review the Industry journals and magazines reporting "best practices"
- Analysts should validate the discovered information with system end-users.

3.8.3 Interviews

- Interviewing the stakeholders involved in the project is a very effective way of gathering the requirements but it is important to interview the right people and also make sure that the interview questions stay focused on o the project relevancy.
- Two kinds of Interviews can be conducted to investigate the requirements - Structured and Unstructured.

1. Unstructured Interview

It is of a question and answer format which is suitable only to obtain general information about the system. This format allows the users to share their needs and ideas.

Advantages	Drawbacks
– Interviewer can gather more and more information as the user is free to answer in his own words. – Interviewer comes to known about new problems and needs during the interview. – Interviewer may come to know about the importance of few areas that were otherwise over looked by them.	– Time consuming as the user just goes on putting his unclear ideas and needs. – Extra information is gathered even which is not needed. – Therefore, analysis of requirements becomes lengthy.

Example : *Unstructured Interview* conducted by an Analyst to a Librarian

Analyst : Hi, I have come to know regarding the functioning of your library.

Librarian : Hello, please come in. I was waiting for you. Will you elaborate about your work?

Librarian : A big problem is managing the identity cards of members. There are so many numbers of cards and many times the cards get misplaced or lost. Then we have to issue a duplicate card instead of it. But it is difficult to find out whether the member is lying or is he genuine.

Analyst : Ok. Then according to you what may be an ideal solution to this?

Librarian : There should be no use of cards at all and all the information should be put into computer which will ease the issue and check how many books are already with a particular member.

Analyst : How often the new members are registered to the library membership?

Librarian : Very often. About 50 to 100 new members are registered in a month. But since two months we have stopped adding new membership because it is already very difficult to manage the existing 500 members. But if the computerized system replaces our manual work then we'll again start registering the new members. From this system, management hopes to earn huge profit.

Analyst : Can you explain me how?

Librarian : See, we get about 50-100 membership requests every month but we are not able to manage it right now. When the new system is built, we will re-open the membership and there is a membership fees to be paid. Management is planning to increase the fee from 400 to 500 for half yearly and 1000 for the whole year. So in this way, we can get huge revenues after automating the system.

Analyst : Do you have different categories in the membership?

Librarian : No, we don't have any categorization for members.

Analyst : How many books do you have in your library?

Librarian : About 10000 books

Analyst : Do you keep records for them?

Librarian : Yes, we do.

Analyst : Do you categorize your books?

Librarian : Yes, by subject.
Analyst : Would you prefer online registration rather than filling up the printed form?
Librarian : Yes, because sometimes we lose these forms and then we don't have any information about that particular member. So, it would be great to computerize the information.
Analyst : Do you have any other requirements from the new system or any other suggestions.
Librarian : It should produce various types of reports and that too faster.
Analyst : What different types of reports do you produce presently?
Librarian : We produce reports for books in the library, reports of the library members, reports about the current suppliers of the books and reports for finance.
Analyst : Can you show me some format of them?
Librarian : Yes, and we want that the same format be used by the new system.
Analyst : Yes, we'll take care of that and any more suggestions?
Librarian : No. we have covered almost all the fields.
Analyst : Thanks for your co-operation and it was nice talking to you.
Librarian : My pleasure. Bye.

Analyst also conducts unstructured interviews of few members of the library in order to know the member's viewpoint and their expectations from the new system :

Analyst : Hello. If you are free, can I ask you few questions.
Member : Sure. About what?
Analyst : Do you know that the library people are planning to automate the library management system?
Member : Yes, I know that.
Analyst : Are you ready to pay little more if there is a computerized system?
Member : Yes, I will if the overall functioning is going to improve and if it helps us in finding the books easily. But by what amount, it should matter.
Analyst : Well as far as I know they are planning to increase the membership fee from 400 to 500 for half year and 1000 for full year.
Member : Uff! That's too much. Then in that case, they should increase the number of books being issued and also the number of days a book can be kept by the member.
Analyst : Ok, then how many books to be allowed for issue and for how many days.
Member : Well should increase number of books from 3 to at least 4 and the number of days for which a book can be kept should be increased from 10 to 15 days. Only then the fee hike will be acceptable.
Analyst : Oh! Yes, they have such plans.
Member : Then it might not bother the members.
Analyst : Ok. And on what basis a search for a particular book can be done?
Member : It must be searched by subject or title.
Analyst : How often you visit this library?
Member : Daily
Analyst : Have you ever recommended this library to your friends, relatives, or to your acquaintances?
Member : Yes, I like this library and I often recommend it to my near ones.
Analyst : Till now, to how many people you have recommended?

Member : May be about 30 people.
Analyst : And how many of them have actually become its members?
Member : 22 of them.
Analyst : That's really nice. Thank You. It was nice talking to you.
Member : Thank You.

After interviewing different people and getting different perceptions about the proposed system, analyst got to know about various types of requirements about the new system.

2. Structured Interview

It is also a question and answer format but it has prescribed answers and the user has to choose the answer from these available choices. It is of either :

- *open response format* where the user is free to answer in his own words such as the question "Why are you dissatisfied with the current system?" or
- *close response format* which limits the users to opt their answer from a set of already prescribed choices such as the question "Are you satisfied with the current system?" or "Do you think that the manual processing be changed with some automated procedure?". The answers to such questions are either 'yes' or 'no'.

Advantages	Drawbacks
– Ensures uniformity in the question types as similar questions with similar choices are asked to the users. Users are not allowed to answer in their own words. Thus, there is uniformity in answering the questions. – Interviewer can easily analyze and interpret the answers as they are chosen from the already prescribed answers. – As similar set of questions are asked, therefore, there is no need of extensive interviewer training. – Less time is needed as the interviews are short because the answers are from the prescribed ones.	– Questionnaire preparation cost is high, – Users most probably do not like to choose the answers; rather they want to put up their needs in more detail and this possible by answering in their own words. – Interviewer is not able to gather requirements in detail.

☞ **Structured v/s Unstructured**

Structured	Unstructured
Less time consuming and the interviewer need not be a trained person	Very much time consuming as the interviewer has no standard set of questions. He interviews the user on the basis of his previous answers and it goes lengthy.
Easier to evaluate objectively since answers obtained are uniform.	High level of structure and mechanical questions are asked.
Users are not allowed to answer in their own words.	Users are free to answer and present their views.

Structured	Unstructured
There is uniformity in answering the questions. Users have to choose the answers from the prescribed choices.	Views are not restricted. So the interviewer gets a bigger area to further explore the issues regarding the system.
Interviewer is not able to gather requirements in detail.	Some issues might come up suddenly while answering some other question.
Interviewer can easily analyze and interpret the answers as they are chosen from the already prescribed answers.	It may happen that the interviewer goes in some undesired direction and the basic facts for which the interview was organized does not get relieved.

☞ **Break up the interview into three steps**

1. **Preparation**
   - Establish the objective for the interview
   - Identify the correct users of the proposed system
   - Identify the project development team members who will participate in the interview.
   - List out the questions and issues to be discussed in the interview
   - Review the related existing documents and systems
   - Set the time and venue to conduct the meeting
   - Inform all the stakeholders about the interview objectives, time and venue.
2. **Enactment i.e. conducting the Interview**
   - List out the issues and error conditions
   - Explore the details by asking more and more questions but relevant to the topic.
   - Take thorough notes
   - Identify and list out the unanswered questions.
3. **Follow-up i.e. making the required changes**
   - Review the interview notes thoroughly for correctness and proper understanding.
   - Write down the information under appropriate models i.e. in proper documents.
   - Identify the areas that need further clarification

3.8.4 Workflows

- Draw the Diagrammatical representation of all the information that is gathered.
- To draw a Sample diagram which can be the representation of the Workflow, follow the below points:
  - Identify agents to create the appropriate swim lanes.
  - Represent steps of workflow with appropriate ovals.
  - Connect activity ovals with arrows to show direction.
  - Use decision symbol to represent either/or situation.
  - Use synchronization bars for parallel paths.

Example : An Activity Diagram to Demonstrate Workflow : Activity diagram to withdraw money from a bank account through ATM.

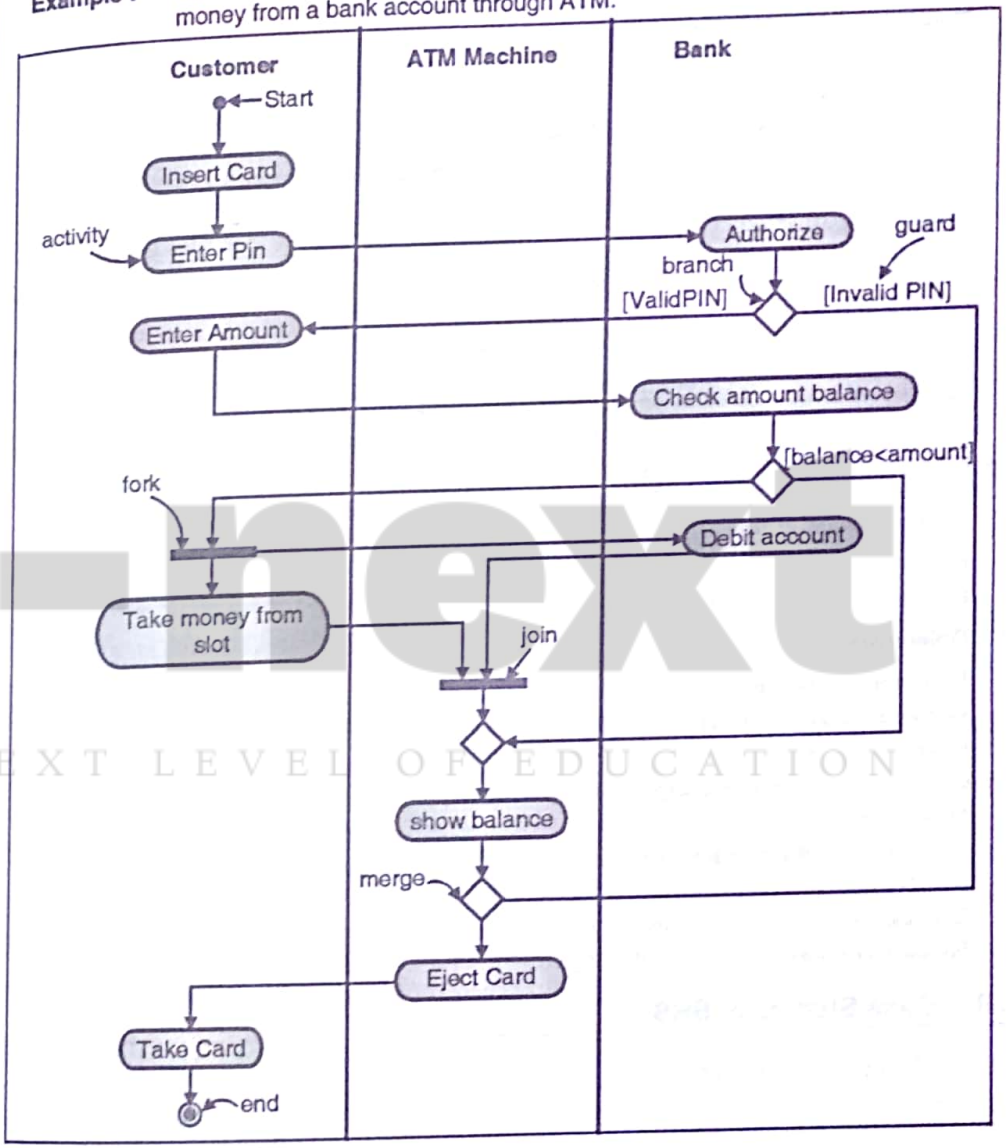


Fig. 3.8.1 : Activity diagram to withdraw money from a bank account through ATM

3.8.5 Building Prototypes

- It is often difficult for the developers as well as for the users to assume or visualize the proposed product when it is still completely new and is yet to be developed. So for the projects which are not

an extension of improvements of an existing project, it is useful to build a model of the system that the end-users and clients can visualize what the final product may look like. And these models are called as Prototypes.

- Prototype is a technique to build a quick and rough version of a desired system.
- Prototypes help to identify the inconsistencies - possible problems and usability issues.
- Prototypes allow users to make design decisions without waiting for the system to be built.
- Prototypes improve the communication between users and developers which lead to fewer changes later and hence reduce overall costs.

Issues with Prototypes

- Designers often try to use patches of code in the real system rather than wasting time in writing the code from the starting.
- Prototypes help in user interface design decisions but they cannot tell you what the original requirements were.
- Designers and end-users focus too much on user interface design and too less on services of the business process which may involve complex database updates and calculations.

3.8.6 Structured Walkthroughs

Tasks of walkthrough

- Reviews findings
- Reviews models based on findings

Objectives

- Find errors and problems
- Ensure that model is correct
- Possible alternatives to custom software development.

Break up the interview into three stages:

1. Preparation:
   - Determine documents to be reviewed
   - Select analyst i.e. the reviewers
2. Enactment i.e. conducting the Interview
3. Follow-up i.e. making the required changes.

3.9 Case Studies of SRS

3.9.1 Online Library System

Q. A library receives 1300 journals of varying periodicals. The journals received have to be recorded and displayed. Action has to be taken journals are not received in time or lost in mail. Unless request for replacement Periodical is sent quickly, it may not be possible to get replacement. Periodicals have to be ordered at different times during the year and subscription renewed in time. Late payment of subscription may lead to non-availability of earlier issues or paying higher amounts for those issues. Current manual systems is not able to meet these requirements. Prepare SRS and systems specification for an information systems for ordering Periodicals.

1. Introduction

Borrowing, returning and viewing the available books at the Library of ITM College is currently done manually where the student has to go to the Library to do the book transactions. Students check the list of available books and borrow it if it is not borrowed by any other else it is wastage of time for the student to come to the library. Then the librarian checks the student id, allows him to check out the book and then updates the member and the books database. This takes at least one to two hours for the member to go to and fro to the library and get his work done.

(a) **Product overview** : The proposed system would be Online Library Management System which would be used by members i.e. either students or professors to check the availability and borrow the books and by the librarian to update the corresponding databases.

The purpose of this SRS is to analyze the needs and features of this proposed *Online Library System* on an high-level.

(b) **Purpose** : The purpose of SRS document is to describe the external behaviour of the Online Library System - operations, interfaces, performance, quality assurance requirements and the design constraints. The SRS includes the complete software requirements for the proposed system.

(c) **Scope :** The *Online Library System* provides the members and employees of the Library with all the books information, online blocking of books and many other facilities. The Online Library System will have the following features.

- The system will be running all day.
- Users can sign-up and login to the system.
- Members can check their account and change their password whenever needed.
- The system allows the members to block the books 24 x 7 hours a day and all through the semester.
- Library staff can check which members have blocked the books and whether they can borrow any more books or not.
- Librarian can create and maintain the books catalog - add/delete books.
- The system updates the billing system whenever a member borrows or returns a book.
- We also have an order department which manages to add or remove a book from the Library.

(d) **Definitions and Abbreviations :**

- ITM – Information Technology and Management
- PIN – Personal Identification Number
- LAN – Local Area Network
- ASP – Active Server Pages
- HTML – Hyper Text Markup Language

(e) **References :**

The *SRS document* uses the following documents as references :

- ITM Information Security Requirements to provide security to the proposed system based on the security system currently used by ITM.
- The Billing System to provide the interface between the proposed system and the billing system currently in use by ITM to update the member account due whenever they borrow and return the books.

2. Overall Description

(a) Product perspective :

The Online Library System is a software package that is useful to improve the efficiency of Libraries, Librarians and Users. The complete overview of the system is as shown in the diagram below. The proposed product has interactions with various kinds of users - Librarian, Members i.e. students and professors of ITM.

The software has to interact with other systems also like: Internet, Billing System and the ITM Information Security System.

(b) Product functions

The Product functions of the system describe the different types of services provided by the system based on the type of users [Member/Librarian].

- The member is provided with the updated information about the books catalog.
- The members can borrow the books they want, if all the other required rules hold correct.
- The member can check his account information and change it any time in the given valid period.
- The members are provided with the books catalog to choose the books which they need.
- The librarian can get the information about the members who have borrowed or returned the books.
- The librarian can add/delete the books available in the book catalog.
- The due to be paid by the member is calculated if in case he is late in submitting the books or his fees is pending and the information along with the due amount about the member is sent to the university billing system.
- The system uses the ITM information security requirements to provide the login facility to the users.

(c) User characteristics

The users of the system are members and librarians of the ITM College and the administrators who maintain the system. The members and the librarian are assumed to have basic knowledge of the computers and Internet browsing. The administrators of the system have more knowledge about the internals of the system because he is responsible to rectify the system in cases of small problems that may arise due to disk crashes, power failures or any other catastrophes.

(d) Constraints

- Information of all the users (members and librarians) must be stored in a database and that must be accessible by the proposed System.
- ITM information security system must be compatible with the Internet applications.
- The Online Library System should be running all 24 x 7 hours a day.
- The users must be able to access the Online Library System from any computer that has Internet connection.
- The billing system is connected to the Online Library System and the database used by the billing system must be compatible the Internet applications.

- The users must be provided with correct usernames and passwords to login to the Online Library System.

(e) Assumptions and dependencies :

- Users have basic knowledge of computers.
- ITM College computer should have Internet connection and Internet server capabilities.
- Users knew English language as the user interface is in English
- The proposed application software can access the college student database.

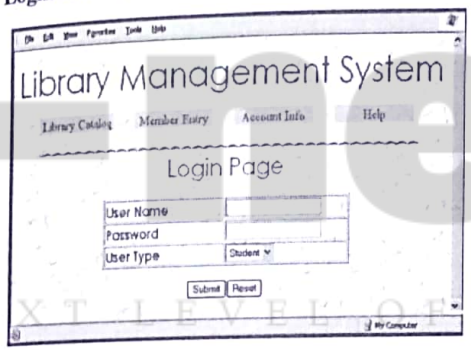
3. Specific requirements

(a) External interfaces :

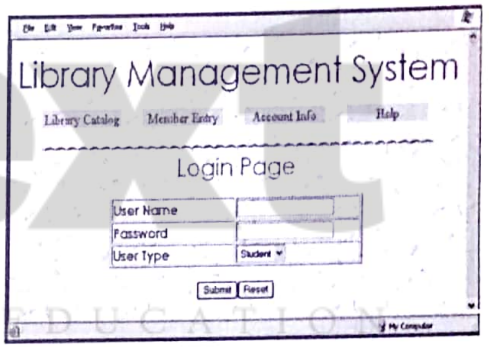
(i) User interfaces : Web Browsers : Microsoft Internet Explorer or Netscape.

The user-interface of the system shall be designed as shown in the user-interface prototypes.

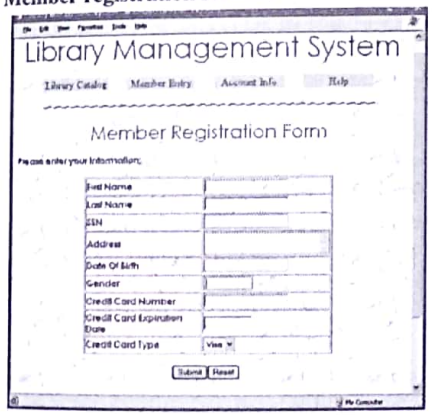
Login screen :



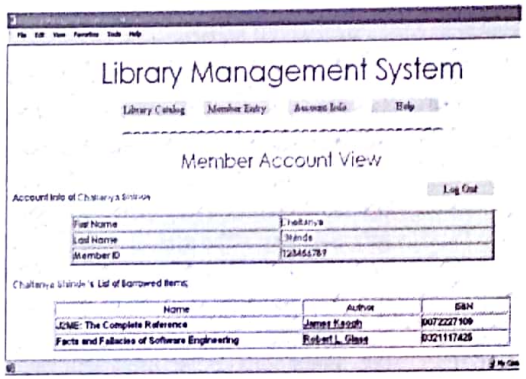
Home page of ITM library:



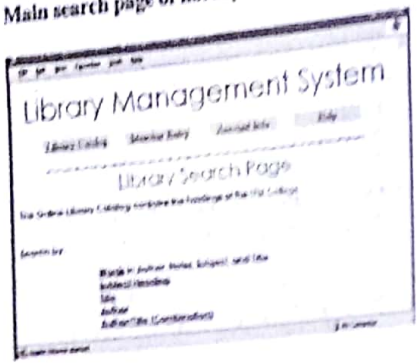
Member registration screen :



Member information once logged in :



Main search page of library catalog :



(ii) **Hardware interfaces :** LAN will be used for collecting data from the users and also for updating the Library Catalogue.

(iii) **Software interfaces :** A firewall will be used with the server to prevent unauthorized access to the system.

(iv) **Communications interfaces :** The Online Library System will be connected to the World Wide Web.

(b) **Design constraints**

- **Programming Language :** The languages that will be used for coding the Online Library System are ASP, HTML, JavaScript, and VBScript. To run ASP pages, the Internet Information Services (IIS) Server needs to be installed.
- **Development Tools :** We will make use of online references for developing the application in ASP, HTML using the two scripting languages - JavaScript and VBScript.

(c) **Functionality**

- **Login Capabilities :** The system shall provide the users with login capabilities.
- **Mobile Devices :** The Online Library System will also be supported on mobile devices such as cell phones.
- **Alerts :** The system will alert the Librarian or the administrator in case of any problems.

(d) **Software system attributes**

(i) **Reliability :** The system has to be very much reliable to avoid the damages to data and prevent from entering incorrect or incomplete data.

(ii) **Availability :** The system is available to the user all 24 × 7 hrs and 365 days a year.

(iii) **Security :** The system shall support the ITM information security requirements and use the same standard as the ITM information security requirements.

(iv) **Maintainability :** The maintenance of the system shall be done as per the maintenance contract.

(v) **Portability :** The users will be able to access the Online Library System from any computer that has internet connection.

(vi) **Performance :**

- **Response Time :** The Information page should be able to be downloaded within seconds using a 56K modem. The information is refreshed every two minutes. The response time for a mobile device must be less than a minute.
- **Throughput :** The number of transactions is dependent on the number of users.
- **Capacity :** The system is capable of handling 200 users at a time.

(e) **Other requirements**

(i) **Licensing requirements :** The usage is restricted to only ITM Library who is purchasing the Online Library System from Library InfoSys and signs the maintenance contract.

(ii) **Applicable standards :** The ISO/IEC 6592 guidelines for the documentation of computer based application systems will be followed.

3.9.2 Purchase Order System

Q. The purchase order system functions as follows : After receiving the purchase requisition from store department, enquiries are made to various suppliers. The suppliers send quotation to the company. All quotation are analysed and final selection of supplier is done and accordingly purchase order to respective supplier are send. The supplier sends invoice along with raw material. Prepare SRS for the above system.

1. Introduction

Store department of a company offers purchase requisition. Upon receiving this requisition, it makes enquiries to various suppliers. Store department then gives specification for stock, name of products and their quantity. Depending on this specification form, suppliers send their quotations of stock quantity and rate of the required products. Depending on the number of suppliers submitting the Quotation, comparative charts are prepared to analyze and then final selection of supplier is done. All these functions are currently done manually which takes a number of days to send the requisition and then receive quotations and then analyze them manually one by one.

(a) **Product Overview**

The proposed system would be Online Purchase Order (OPO) system which would be used by company staff and various suppliers to offer the purchase requisition and send the corresponding quotations, also computationally analyze the quotations so as to select the appropriate suppliers.

(b) **Purpose**

The purpose of this document is to describe the external behaviour of the Store Departments, description of products, including handling persons, product perspective, overview of requirements, general constraints. It will also provide the specific requirements and functionality needed for this system.

(c) **Scope**

The purchase order system assists the company to purchase product according to their needs and specification at minimum rate and in minimum time so that it fulfills storekeeper's needs and specifications. The availability of the product will be fulfilled before the shortage and requirements rose. The OPO System will have the following features :

- Stock Maintenance
- Regular product master
- Payment and receipt maintenance according to date and time specification.
- Report Generation (weekly, Monthly, Yearly) of various details such as :

(i) Total stock of products
(ii) Total Form collection (total number of applied suppliers)
(iii) List of products expired.

(d) Definition, Acronyms and Abbreviation

- OPOS - Online Purchase Order System
- ITM – Information Technology Management

(e) References

The *SRS document* uses the following documents as references :

- Store requisition form and supplier's quotation form as a sample format so as to de the form in the same format when the current system is automated.
- Information Security Requirements to provide security to the proposed system base the security system currently used by the company.
- The Billing System to provide the interface between the proposed system and the bill system currently in use by store to update the product information whenever a p product is purchased or ordered (sold).

2. Overall Description

(a) Product Functions

- OPOS generates Quotation form for different supplier for same product requirements.
- OPOS will be automated and centralized hence it will create a comparative chart for sam product specification but different rating.
- Also that chart have to show sorting of rates through which we can easily analyze th suppliers.
- OPOS keeps the record of the product code, product name, quantity and the oth specification.

(b) User Characteristics

System user is storekeeper/ Store Manager/suppliers. The users have elementary compute knowledge.

(c) Design Constraints

- OPOS requires a computer equipped with 133 MHZ Intel Pentium Processor, with minimum 64/a24 MB RAM, a CPU speed of 200 MHZ or above for good performance.
- At least WIN OS should be there.
- Minimum 64 MB RAM recommended.
- ORACLE server should be installed.
- Minimum DOT MATRIX PRINTER should be installed.

(d) Assumptions and dependencies

- Users have basic knowledge of computers.
- Company's computers should have Internet connection and Internet server capabilities.
- Users knew English language as the user interface is in English
- The proposed application software can access the company product database.

3. Specific Requirements

(a) User interface

All the forms are GUI based and are used by user and manager interactively except the Quotation form.

(b) Database Names

- Product Master
- Stock Mater
- Concession Rule Details
- Tax Details
- Suppliers' Norms
- Departure details

(c) Functional Requirements

Determine the last date and time for the filling of Quotation form and constraints are satisfied.

- No more than one Quotation form should be filled at the same time for same product specification.
- The supplier should follow the whatever stock requirement specification.
- Preferences is given to discount offering and quality maintained suppliers.
- The supplier should follow rules an constraint as defined, in any manner they does not violate these constraints.
- Allow the user to maintain separate form for each supplier.
- Allow the user to maintain a database of products, quantity.
- To maintain a list of all product and stock specification available in store department.
- To generate report for number of suppliers, supplying number of products, quality details, rates, discount and tax calculation, date of supplying after receiving order etc.

(d) Software System Attributes

- The generated problems should be solvable.
- The database should be properly synchronized with record generated.

(i) **Reliability :** The system has to be very much reliable to avoid the damages to data and prevent from entering incorrect or incomplete data.
(ii) **Availability :** The system is available to the user all 24 × 7 hrs and 365 days a year.
(iii) **Security :** The system shall support the ITM information security requirements and use the same standard as the ITM information security requirements.

(iv) **Maintainability :** The maintenance of the system shall be done as per the maintenance contract.

(v) **Portability :** The users will be able to access the OPOS from any computer that has Internet connection.

(vi) Performance :

- **Response Time :** The Information page should be able to be downloaded within seconds using a 56K modem. The information is refreshed every two minutes. The response time for a mobile device must be less than a minute.
- **Throughput :** The number of transactions is dependent on the number of users.
- **Capacity :** The system is capable of handling 200 users at a time.

4. Acceptance Criteria

Before accepting the system, the developer must demonstrate that the system works on the number of stock data, product, quantity specification. The developer will have to show through test cases that all conditions are satisfied.

3.9.3 Hospital Management System

1. INTRODUCTION

➢ 1.1 PRODUCT OVERVIEW

- This Software Requirements Specification formally specifies Hospital Patient Management System (HPMS). It includes the resulting decisions of both - business analysis and systems analysis efforts. The objective of this document is to describe the system's high level requirements such as functional requirements, non-functional requirements and business rules and constraints.
- The detail structure of this document *is organized* as follows:

  Section 2 of this document provides overview of the business domain functions that the proposed Hospital Patient Management System (HPMS) will support.

  Section 3 presents detail requirements and overview of the Hospital Patient Management System's functions.

➢ 1.2 PURPOSE

- The purpose of this SRS document is to describe the external behaviour of the Hospital Patient Management System (HPMS) - operations, interfaces, performance, quality assurance requirements and the design constraints.
- Developers should refer to this document as the only source of requirements for the project. They should not consider any requirements written or verbal as valid until they appear in this SRS document or in its revised version.

➢ 1.3 SCOPE

- The system will allocate beds to patients on priority basis, and assigns doctors to patients in the designated wards as needed.
- Doctors will use the system to keep track of the patients assigned to them.
- Nurses will use the system to keep track of available beds, patients, and the type of medication given and to be required for each patient.

- The current system in use is a paper-based system which cannot provide updated lists of patients within a reasonable timeframe. Therefore, the aim of the system is to reduce over-time pay and increase the number of patients that can be treated accurately.

➢ 1.4 AUDIENCE

The intended audience include all stakeholders such as administrative staff, doctors, nurses, surgeons and developers.

i. *Front-desk staff:* They are responsible for patient's check-in or notification of appropriate people (e.g. notify administrator or nurse when an event occurs).

ii. *Administrators*: Every administrator has basic computer knowledge. They are responsible for all the scheduling and updating the day/night employee shifts; and assigning doctors and nurses to patients.

iii. *Nurses*: They are responsible for assigning patients to appropriate wards if the beds are available, otherwise putting patients on the waiting list.

iv. *Doctors and Surgeons*: They will use the HPMS to check their patient's list and their duty schedule.

➢ 1.5 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

HPMS	Hospital Patient Management System
PHN	Personal Health Number on health card
Report	an account of patients
Database	collection of information in a structured form
Front-desk staff	administrative staff at reception desk
Logon ID	a user identification number to enter the system
Password	a word that authenticates a user to the system
Web-based application	an application that runs on Internet
ID	Patient Identification number
GUI	Graphical User Interface
SRS	Software Requirements Specification

2. OVERALL DESCRIPTION

➢ 2.1 PRODUCT PERSPECTIVE

This HPMS is a self-contained system that manages at most activities of the hospital such as bed assignment, operations scheduling, personnel management and administrative issues.

➢ 2.2 PRODUCT FUNCTIONALITY

The system functions are as follows:

i. *Registration :* When a patient is admitted, the front-desk staff checks whether the patient is already registered with the hospital. If so, his/her PHN is entered into the computer else a new PHN is given to this patient and his information including his date of birth, address and contact number is entered into the system.

ii. *Consultation :* The patient then goes to consultation-desk to explain his/her condition so that the consulting nurse can decide what kind of ward should be assigned to him/her. There are two possible situations :

If a bed is available in the ward, then the patient is allotted that bed and asked to wait for the doctor to come.

If there is no bed, the patient is put on a waiting list until a bed becomes available.

iii. *Check out* : When a patient checks out, the front-desk staff shall delete that patient's PHN from the system and the just vacate bed is included in available-beds list.

iv. *Report Generation* : The system generates reports of the following information: patients, bed availability and staff schedules after every six hours.

2.3 USER CHARACTERISTICS

The system will be used by the hospital staff who include administrators, doctors, nurses and front-desk staff who will be trained on using the system. The system is also designed to be user-friendly by making use of a Graphical User Interface (GUI).

2.4 OPERATING ENVIRONMENT

i. The system will use MySQL Database which is open source and free.

ii. The Development environment will be Windows XP SP2.

iii. The system will be a Web-based application.

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

- The system must be delivered by August 25th 2012.
- The existing Telecommunication infrastructure of HPMS is based on IEEE100802 standards and the system must confirm to this standard.

2.6 ASSUMPTIONS AND DEPENDENCIES

i. It is assumed that 100 IBM compatible computers will be available before the system is installed and tested.

ii. It is assumed that the hospital will have enough trained staff to take care of the system.

3. SPECIFIC REQUIREMENTS

3.1 FUNCTIONAL REQUIREMENTS

i. Registration

- Add patients : The front-desk staff will use HPMS to add new patients to the system.
- Assign ID : The front-desk staff will use HPMS to give each patient an ID and add it to the patient's record.

ii. Consultation

- Assign Ward : The consulting nurse will use HPMS to assign the patient to an appropriate ward.
- Assign to Waiting List : The consulting nurse will use HPMS to assign Patient to a waiting list if no bed is available.

iii. Medical Management

- Assign Doctor : The administrative staff will use HPMS to assign a doctor to a given patient.
- Assign Nurse : The administration staff will use HPMS to assign a nurse to a given patient.
- Inform Doctors : The HPMS will inform doctors about the new patients.
- Inform Nurses : The HPMS will inform the nurses about new patients.

- Emergency Case : In case of emergency, the administrative staff will use HPMS to assign an emergency room, doctors and nurses immediately to the patient.
- Surgery case : In case of surgery, the administrative staff will use HPMS to assign a surgery room, surgeon and nurses to the patient.
- Generate Report : The HPMS will generate the patient's situation record every two hours for normal patients, and for every half hour for severe patients.
- Record Procedure : The complete statement procedure for the patient will be recorded by the system.
- Inform Patient : The HPMS will automatically inform about the patients who are on waiting list and about available beds whenever they become available.

iv. Check Out

- Delete Patient ID : The administrative staff will use HPMS to delete the ID of the patient from the system when the patient checks out.
- Add to beds-available list : The administrative staff allowed to put the beds just evacuated in beds-available list.

v. Report Generation

- Patient information : Every six hours the HPMS will generate reports about the following information - patient's PHN, patient's name, ward name, bed number and the doctor's name.
- Bed Availability : Every six hours the HPMS will generate report on bed availability that includes ward name, bed number, occupied/unoccupied.
- Staff Schedule : Every six hours the HPMS will generate report on staff schedule that includes staff ID, staff name, staff type, duty shift.

vi. Database

- Patient Mandatory Information : first name, last name, phone number, personal health number, address, postal code, city, country, patient identification number.
- Update Patient Information : The HPMS shall allow the user to update any of the patient's above information.
- Search for Patient : The HPMS will allow user to search for patient's information by last name or PHN or patient ID.
- Staff Mandatory Information : identification number, first name, last name, phone number, address, postal code, city, country, employee type, duty schedule.
- Update Staff Information : The HPMS will allow user to update any of the staff's information described above.
- Search Employee Information : The HPMS will allow user to search for employee information by last name, or ID number.
- Ward Types : Maternity, Surgical, Cancer and Cardiac.
- Ward Information : ward name, ward number, list of rooms in ward.
- Room Information : room number, list of beds in room, full/not full.
- Bed Information : bed number, occupied/unoccupied, patient PHN.
- Ward Search : The HPMS will allow users to search the ward, room, and bed directly by ward number, room number and bed number respectively.

3.2 BEHAVIOUR REQUIREMENTS

➢ 3.2.1 USE CASE VIEW

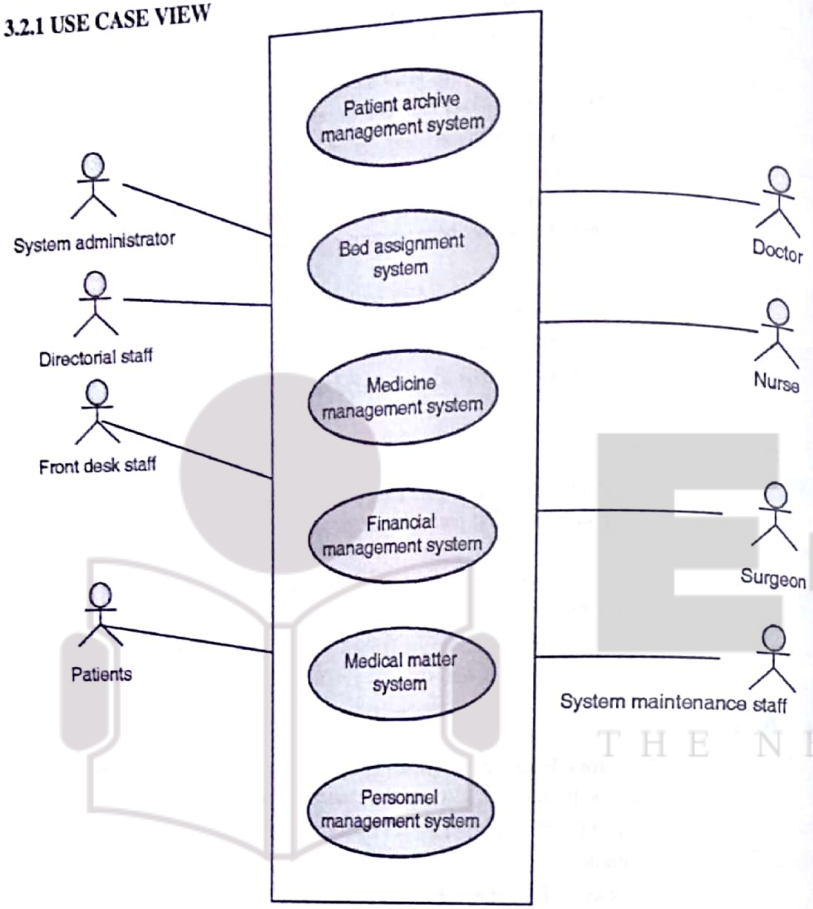


Fig. 3.9.1

4. OTHER NON FUNCTIONAL REQUIREMENTS

➢ 4.1 RELIABILITY

The system has to be very much reliable to avoid the damages to data and prevent from entering incorrect or incomplete data.

➢ 4.2 AVAILABILITY

The system is available to the user all 24 × 7 hrs and 365 days a year.

➢ 4.3 SAFETY and SECURITY

- Patient Identification : The system identifies the patient using PHN.
- Login ID : Any user who uses the system shall have a Login ID and Password.

- Modification : Any modification such as insert, delete, update for the Database shall be allowed only to the administration staff.
- Compliance : The system must comply with the Regional Health Authority Regulations.
- Front Desk staff Rights : Front Desk staff can view and add all information in HPMS, but shall not be able to modify any information in it.
- Administrators' Rights : Administrators can view and modify all information in HPMS.
- Nurses' Rights : Nurses can only view the information in HPMS.
- Doctors Rights : Doctors can view all information in HPMS.

➢ 4.4 MAINTAINABILITY

- Back Up : The system will provide back-up capability to the Data.
- Errors : The system shall keep a log of all types of errors.

➢ 4.5 PERFORMANCE

- Response Time: The system will give responses in a second after checking the patient's information.
- Capacity : The System will support 1000 people at a time.
- User-interface : The GUI will respond within 5 seconds.
- Conformity : The systems must conform to the Microsoft Accessibility guidelines.

3.9.4 Catering System

Example 3.9.1 :

Joshi Caterers Pvt. Ltd. wants to develop the order processing and billing software which presently works as under:

Company collects order from different corporate customers or individuals. The customer fills up the order form describing various details like order details, customer details, menu item details and number of thalies. 50% advance is collected only from individual customers. Then after receiving the orders, Kitchen Order Ticket (KOT), is issued to the kitchen and then kitchen issues the list of raw material (excluded from available stock) to be purchased from the suppliers. Purchase order is given and ordered material is received from the fixed suppliers and forwarded further to the kitchen. After completion of the delivery of the order, bill is issued to the customer on the basis of actual number of thalies or ordered number of thalies whichever is more. Payment is accepted and receipt is given to the customer. Prepare SRS and system specification for the above system.

Soln. :

1. INTRODUCTION

➢ 1.1 PRODUCT OVERVIEW

The objective of the new system is to provide scalability, reliability and efficiency to enable productivity increases over the ordering and invoicing process resulting in reduced administration and processing costs.

➢ 1.2 PURPOSE

This Software Requirements Specification formally specifies Joshi Caterers Pvt. Ltd. (JCPL). It includes the resulting decisions of both - business analysis and systems analysis efforts. The

purpose of this document is to specify the software requirements for JCPL and provide an ongoing reference for project staff.

1.3 SCOPE

The JCPL system will allow the following functionalities:

i. *Collect order:* customer fills up the order form describing various details like order details, customer details, menu item details and number of thalies
ii. *Advance Payment* : 50% advance is collected only from individual customers.
iii. *Issue KOT* : After receiving the orders, Kitchen Order Ticket (KOT), is issued to the kitchen.
iv. *Check Stock Level* : kitchen issues the list of Utensils and Food Items below minimum stock level
v. *Purchase Order* : Purchase order is given and ordered material is received from the fixed suppliers and forwarded further to the kitchen.
vi. *Issue bill* : After completion of the delivery of the order, bill is issued to the customer on the basis of actual number of thalies or ordered number of thalies whichever is more.
vii. *Payment* : customer makes payment and gets receipt.
viii. *Total Income* : income generated from services annually

1.4 AUDIENCE

The intended audiences include administrative staff, customers and developers.

i. *administrative staff:* Every administrator has basic computer knowledge. They are responsible for collecting orders, informing the suppliers for raw material and collecting payment from the customers.

1.5 DEFINITIONS and ABBREVATIONS

- *Acceptance* – the date on which the contract is made.
- *Change Database* - manages all change proposals and this is linked to a version management system.
- *Critical Failure* - failure that results in System's data not being available to more than 3 users.
- *Data Recovery* - restoring data that has been physically damaged.
- *Fault Tolerance* - a technique that ensures that system errors do not result in system failures.
- *Form Editor* – allows change proposal forms to be filled in once again.
- *Interface Generator* - graphical screen design system where interface components such as menus, field, icons and buttons are selected from a tool box and positioned on the interface.
- *System Availability* –total number of hours the system was Operational divided by the total number of hours the system was under a critical failure.
- *JCPL* - Joshi Caterers Private Limited.
- *KOT* - Kitchen Order Ticket
- *CM* - Configuration Management a standard process involving the application of predetermined procedures. They require careful management of very large amounts of data and attention to detail is essential.

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

The proposed system would be Online Catering Management System which would be used for order processing and billing.

As indicated in below figure, interaction between the subsystems will be via the underlying data stores. Each of the subsystems in self-contained, hence, there is no direct interaction required between any of the subsystems.

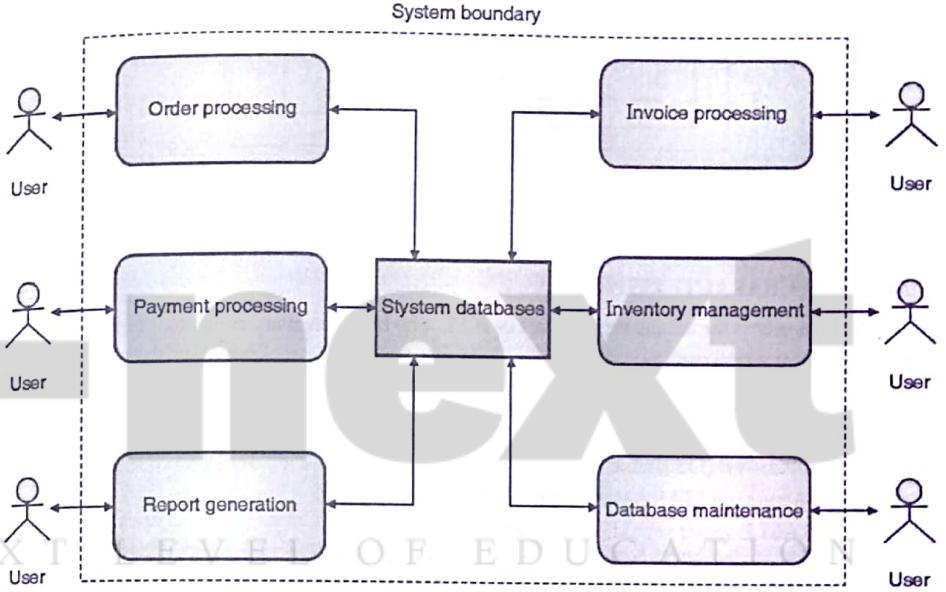


Fig. P. 3.9.1

2.2 PRODUCT FUNCTIONALITY

- JCPL collects order from different corporate customers or individuals.
- JCPL stores various details like order details, customer details, menu item details and number of thalies.
- JCPL generates KOT which is then issued to the kitchen.
- JCPL finds the list of Utensils and Food Items below minimum stock level.
- JCPL places purchase order to its suppliers giving the list of requirement.
- JCPL generates catering bill and issues acknowledgement after delivering the order to the customer.

All the above functions can be summarized through the following context level diagram;

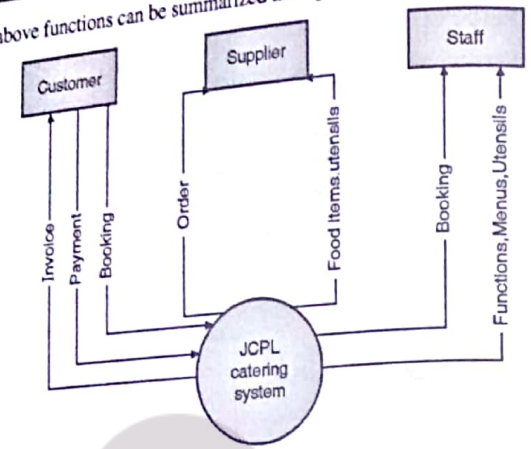


Fig. P. 3.9.1(a)

2.3 USER CHARACTERISTICS

System user is the administrative staff of JCPL who has elementary computer knowledge.

2.4 DESIGN CONSTRAINTS

- JCPL requires a computer equipped with 133 MHZ Intel Pentium Processor, with a minimum 64/a24 MB RAM, a CPU speed of 200 MHZ or above for good performance.
- At least WIN OS should be there.
- Minimum 64 MB RAM recommended.
- ORACLE server should be installed.
- Minimum DOT MATRIX PRINTER should be installed.

2.5 ASSUMPTIONS AND DEPENDENCIES

- Users have basic knowledge of computers.
- Users knew English language as the user interface is in English
- The proposed application software can access the company product database.

2.6 USER DOCUMENTATION

The System shall be delivered to JCPL attached with all of the following documents:

- A complete online help facility that will assist system users to easily operate the system.
- Electronic user manual on CD-ROM in Adobe Acrobat format which will be useful for initial training of System users.
- Instruction manual for System Administrator on CD-ROM in Adobe Acrobat format which will be useful for initial training of System Administrators.

3. SPECIFIC REQUIREMENTS

3.1 USER INTERFACE

All the forms are GUI based.

3.2 DATABASE NAMES

- Menu Master
- Stock Mater
- Order Master
- Concession/Discount Rule Details
- Suppliers Details
- Payment Details
- Customer Details

3.3 FUNCTIONAL REQUIREMENTS

i) Customer Details

a) JCPL enables addition, modification, deletion, display and storage of information about customers such as a unique identifier, name, address, contact number and E-mail address (optional).

b) JCPL avoids redundancy – while the user is adding custoemr details, if the customer's name and address match an existing customer entry, the System will reject the new information and tells the user that the customer already exists.

ii) Supplier Details

a) JCPL enables addition, modification, deletion, display and storage of information about suppliers such as a unique identifier, name, address, contact number and E-mail address (optional).

iii) Food Item and Utensil Details

a) JCPL enables addition, modification, deletion, display and storage of information about food items and utensils.

b) JCPL stores following information about food item and utensils - a unique identifier, Food Item and Utensil Name, suppliers who provide this food items and utensils, and per-unit price that each supplier charges for the food item and utensil.

c) JCPL provides following details about below minimum level of food items and utensils in stock :

- The number of units of the food items and utensils currently in stock
- The number of units of the food items and utensils currently on order
- The price per unit of the food item and utensil
- The stock reorder level or minimum stock level

iv) Service Details

a) JCPL enables addition, modification, deletion, display and storage of information about services.

b) JCPL provides following details about each service:

- A unique identifier
- Service Type
- The number of food dishes on ordered menu
- The number of utensils required for the service

- The price per service

v) **Booking Details**

a) JCPL enables addition, modification, deletion, display and storage of information about bookings.

b) JCPL will provide following information about each booking:
- A unique identifier
- booking date
- the customer that made the booking
- The quantity and type of each service booked
- The invoice associated with the booking

c) JCPL stores 50% advance payment details for a booking if the customer is an individual person.

d) JCPL generates KOT after accepting the menu order from the customer. This KOT is then issued to the kitchen.

vi) **Invoice Details**

a) JCPL generates catering bill/invoice on the basis of actual number of thalies or ordered number of thalies whichever is more.

b) JCPL stores following information for each invoice:
- A unique identifier
- Invoice date – the date that the invoice is entered
- The customer to whom the invoice applies
- The services that make up the invoice

vii) **Payment Details**

a) JCPL enables addition, modification, deletion, display and storage of information about payments.

b) JCPL provides following information for each payment:
- A unique identifier
- Payment date
- The customer making the payment
- The invoice against which the payment is being done.
- The amount paid

viii) **Unique Identifier Selection**

JCPL shall automatically assign and display a unique numerical identifier when a new instance of any of the following objects is created :
- A customer,
- A supplier,
- A menu,
- A company,
- A food item,
- A menu1,

- An order,
- A payment, or
- An invoice.

ix) **Report Generation**

a) JCPL shall display a requested report at the user's data display device by sorting the information according to the preference specified by the user.

b) JCPL shall include the following information at the start of each report:
- The date and time at which the report was produced, and
- The title of the report.

A) **Minimum Stock Level Report**

a) When requested by a user, JCPL system shall automatically generates a minimum stock level report for all food items and utensils where the quantity in stock is below the stipulated minimum level for that food item or utensil.

b) JCPL system shall display at least the following information when a minimum stock level report has been generated:
- The nominated minimum storage level for the food item or utensil
- The actual storage level for the food item or utensil.

B) **Customer Report**

a) JCPL System shall automatically generate customer monthly service report that lists all details of customer services during the last month.

C) **Nil Payment Report**

a) JCPL System shall automatically generate nil payment report that lists all details of all invoices raised in the previous month against which no payments have been made.

D) **Total Income Report**

a) JCPL System shall automatically generate a total income report that calculates the total income generated from functions and services during the whole annual year.

4. OTHER NON FUNCTIONAL REQUIREMENTS

4.1 PERFORMANCE

- The System shall display the latest information stored in the System at the time of any user request.
- The System shall respond to all user commands within an average time of 5 seconds of a request being made.
- The System shall complete all users commands and display the result within an average time of 15 seconds of a request being made.

4.2 PORTABILITY

The System shall be able to run under any combination of each of the following operating systems:

a) Windows XP b) Windows 2000,
c) Solaris d) Unix.

4.3 SECURITY

The System shall provide appropriate facilities to ensure that only authorised users have access to the information stored in the System.

Review Questions

Q. 1 What is SRS ? Explain need & benefits of SRS.

Q. 2 Explain the issues of requirement gathering ?

Q. 3 What are the techniques of requirement gathering ?

Q. 4 What is Requirements engineering?

Q. 5 Describe the requirements engineering tasks?

Q. 6 What is the need of Requirements' Validation ?

Q. 8 Explain Requirement Management.

Q. 9 List the various stakeholders involved in requirement analysis ?

Q. 10 Write an SRS for Maharashtra State Electricity Board. Assume the functional and non-functional requirements.

□□□

UNIT I

CHAPTER 4

Object-oriented Design using UML

Syllabus

Class diagram, Object diagram, Use case diagram, Sequence diagram, Collaboration diagram, State chart diagram, Activity diagram, Component diagram, Deployment diagram

4.1 Introduction

- Unified Modeling Language (UML) begins by constructing a model which is a simplification of reality.
- Model is an abstraction of some underlying problem.
- Model is a complete description of a system from any particular perspective constructed to understand the system before building the new or before modifying the existing system.

Static and Dynamic Models

- A *static model* is the snapshot of system's parameters at rest or at a specific point of time. For example, a customer could have more than one account at a time. Static models have stability and there is no change of data in future. The UML *class diagram* is a type of a static model.
- A *dynamic model* is a collection of behaviours of the system. It represents how objects interact with each other in order to perform any task. The UML *interactive* (*sequence and collaborative*) and *activity diagrams* are types of a dynamic model.

Benefits of a Model

1. Clarity

Models make it easier to. We can very easily express complex ideas using models as it allows visual examination of the whole system possible.

2. Complexity

Models reduce complexity by separating unimportant aspects from those that are important.

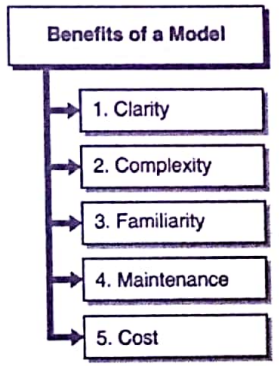


Fig. C4.1 : Benefits of a Model

➜ **3. Familiarity**

Models are the representation of how the information is actually represented so that you may understand the system and feel more comfortable to work.

➜ **4. Maintenance**

Visual identification and confirmation about the locations to be changed will reduce errors. Also manipulation (changing) of a model is more easier than manipulating a real system.

➜ **5. Cost :**

The cost of building and modifying a model is much lower as compared to the same task performed on a real system.

4.2 UML

- UML is a set of notations and conventions used to model an application to describe system's parameters and behaviours..
- The UML is an object-oriented methodology and technique because it is generally applicable to object-oriented problem solving.
- UML is not defined as 'data modelling' technique, but as an "object modelling" technique. Instead of entities, it models "object classes".
- Because of confluence in ideas, techniques, personalities, and politics; UML became a standard notation for representing the structure of data in the object-oriented community. It was developed when the three great personalities of the object-oriented community - James Rumbaugh, Grady Booch, and Ivar Jacobson agreed to adopt a standard notation for UML.
- UML was originally developed at Rational Software but is now administered by Object Management Group (OMG) i.e. responsible for creating and maintaining the UML language specifications.

OMG defines UML as, "a graphical language" for :

- **Visualizing :** There are some aspects of software system that you can not understand unless you build models.
- **Specifying :** models are precise, unambiguous, and complete.
- **Constructing :** allows direct connection to a variety of programming languages.
- **Documenting :** artifacts of a software system.

4.2.1 Features of UML

➜ **1. Syntax**

UML is just a language that tells what model elements and diagrams are available and the rules associated with them. It doesn't tell you what diagrams to create.

➜ **2. Comprehensive**

It can be used to model anything that may fill the user's requirement.

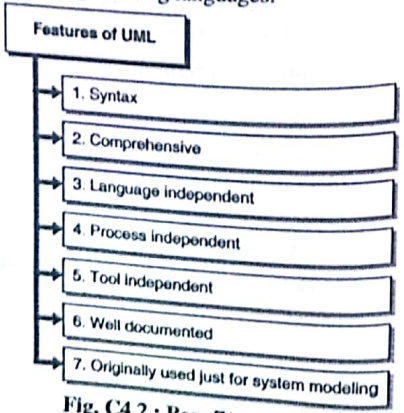


Fig. C4.2 : Benefits of a Model

➜ **3. Language independent**

It does not matter what high-level language is to be used in the code. Mapping into code is the responsibility of the case tool that you use.

➜ **4. Process independent**

The process by which the models are created is separate from the definition of the language.

➜ **5. Tool independent**

UML gives freedom for tool vendors to be creative and add value to visual modeling with UML.

➜ **6. Well documented**

The UML notation guide is available as a reference to all the syntax available in UML language.

➜ **7. Originally used just for system modeling**

Some user defined extensions are becoming more widely used now, for example for business modeling and web-based modeling.

4.2.2 Need of UML

- **Software construction needs a plan :** Software development is a very complex job and once a particular structure is in place, it is very difficult job to make any changes. UML helps in constructing a visual model of the real system.

  Example :A building architect creates a drawing of the building from more than one direction, ensuring that the structure and dimensions of the building are appropriate. Only when this visual model of the proposed work is approved, then the real construction of the building i.e. laying of bricks is begun.
- **Visualize in multiple dimensions and levels of detail :** As in the above example, UML, allows software to be visualized from multiple dimensions so that a computer system can be completely understood before construction begins.
- UML is also used to produce several models at increasing levels of detail. These increasing levels of detail can be added to each part of the software as it is constructed until final stage.
- **Appropriate for both new and legacy systems :** UML can be applicable for both new system developments and improvements in existing systems where only those parts that need modifications are modeled.
- **Documents software assets :** Undocumented or poorly documented software has a very low value as a company asset. Documentation saves the company from losses occurred by depending on key personnel who may leave at any time and thus improves understanding of the company's critical business processes.
- **Unified and universal :** OMG has made UML as a de-facto (actual) standard modeling language in the software industry. UML is called as a universal language because it can be applied in many areas of software development.
- **Inherent traceability :** The Use Case driven approach of UML modeling ensures that all levels of model trace back to elements of original functional requirements.
- **Incremental development and re-development :** UML models can be applicable in incremental development environment. It is not only possible to develop only those parts of the model that are required to satisfy the new requirements, but also possible to demonstrate that the code needed to fulfil these requirements is in place.
- **Parallel development of large systems :** Large complex UML models can be decomposed into simple models that can be developed independently by different people or groups simultaneously at the same time.

4.2.3 UML Advantages

- UML modelling is effective for large, complex software systems and also for modeling middleware.
- It is easy to learn and provides advanced features for expert analysts, designers and architects.
- It can specify the system in an implementation-independent manner.
- It allows re-usability i.e. 10-20% of the constructs are used 80-90% of the times.
- Structural UML modeling represents a skeleton that can be refined and extended with additional structure and behavior.
- Use case modeling specifies the functional requirements of the system in an object-oriented manner.
- Allows modelling of any type of application m running on any type of combination of hardware, operating system, programming language, and network.
- Increases efficiency and thus reduces cost and time-to-market.
- UML Profiles (i.e. subsets of UML designed for specific purposes) help to model Transactional, Real-time, and Fault-Tolerant systems in a natural way.

4.2.4 UML Diagrams

The most common 8 UML diagrams are :

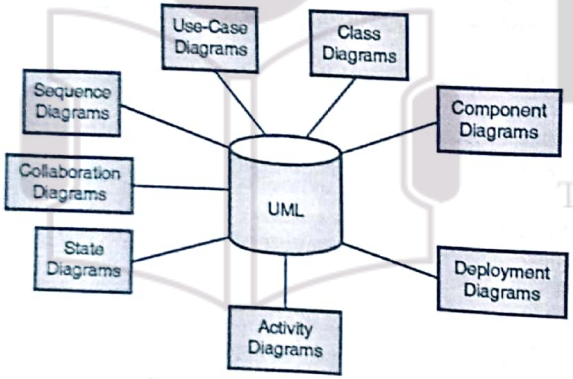


Fig. 4.2.1 : UML diagrams

Table 4.2.1 : Classification of Key UML Diagrams

1.	System Requirements	1. Use Case Diagram
2.	System Structure	2. Class Diagram 3. Interactive/Communication/Collaborative Diagram 4. Component Diagram 5. Deployment Diagram
3.	System Behavior	6. Interactive Sequence Diagram 7. State Diagram 8. Activity Diagram

Syllabus Topic : Use-case Diagrams

4.3 Use-case Diagrams

- Use case diagrams describe what a system does from an external observer's viewpoint. The focus is on **what** a system does rather than **how**.
- Use case diagrams are based on scenarios. A ***scenario*** is an example of ***what*** happens when someone interacts with the system i.e. it describes the course of events that may take place in the system.
- A Use Case Diagram shows relationships between ***actors, use cases*** and their ***associations*** (also called as interactions or communications) in a ***system***.
- Actors are represented using *stick figures,* Use cases by *ovals* and Communications by *lines* that link actors to use cases.
- A use case diagram is a visual representation of different *scenarios* showing the *association* between an *actor* (primary elements/business roles) and a *use case* (business processes that forms the system).

Example : A use case scenario for a medical clinic:

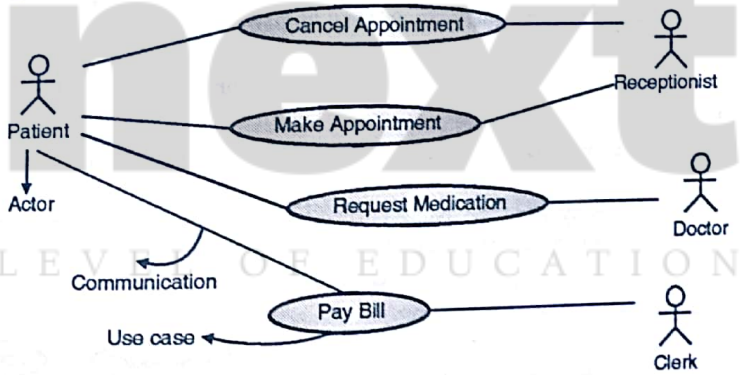


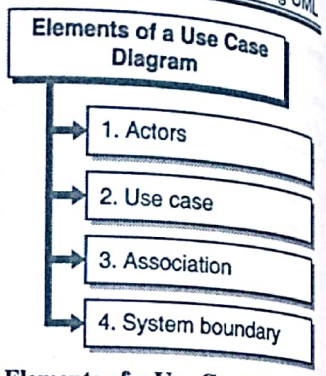
Fig. 4.3.1 : Use case diagram for a medical clinic

A patient calls-up in a clinic to get an appointment for monthly check-up. The receptionist finds out the nearest empty time slot in the appointment book and schedules the appointment for that time slot.

4.3.1 Need of Use Case Diagrams

1. **Determines requirements :** New use cases often generate new requirements as the system is analyzed and so on, the design takes shape. It is helpful in project planning, documentation and development of system's requirements
2. **Communicates with clients :** Notations in use case diagrams help developers to easily communicate and discuss with clients.
3. **Generates test cases :** Collection of scenarios in a use case leads to set of test cases for those particular scenarios.
4. **Represents complete functionalities :** Use cases represent complete functionality of the system or say, business process rules.
5. Discovers classes and relationships among subsystems of the system.

4.3.2 Elements of a Use Case Diagram



Use case diagram constitutes of following elements :

1. Actors

- An actor in a use case diagram is any entity that performs certain roles in a given system. These may be the business roles of users who perform certain functionalities in a given system. If some entity does not make any affect on a certain piece of functionality then, it makes no sense to represent it as an actor.

Fig. C4.3 : Elements of a Use Case Diagram

- An ***actor*** is who or what initiates the events involved in that task.
- Identify an actor in the proposed system by searching for business terms that represent certain roles.
- An actor is shown as a stick figure in a use case diagram portrayed outside the system boundary.

Example : In the statement "students visit the library to borrow books. Librarian updates the catalogue" - *student* and *librarian* are the roles that are identified as actors in the system.

Fig. 4.3.2 : Actors in a Library Use-case diagram

2. Use case

- A use case in a use case diagram is a visual representation of distinct business functionality in a given system.
- Identify the use cases by listing the unique business functions in your problem statement. Each of these business functions can be classified as a potential use case.
- A use case is shown as an ellipse in a use case diagram.

Example : In the same above statement about library, there are two functions - "Borrow book" and "Update catalog" that are identified as use-cases in the system.

Fig. 4.3.3 : Use cases in a Library Use case diagram

3. Association

- An association is an interaction between an actor and a use case.
- Associations between actors and use cases are indicated in use case diagrams by solid lines.
- Associations are shown by lines connecting use cases and actors to one another, with an arrowhead (arrowheads are very rarely used) on one end of the line. The arrowhead is used only when you need to indicate the direction of initial invocation or to indicate the primary actor within the use case.

Example : In the same above statement about library, there is an association between actor 'student' and use-case 'borrow book'.

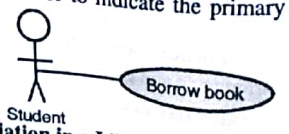


Fig. 4.3.4 : Example of an Association in a Library use case diagram

4. System boundary

- A system boundary of a use case diagram defines the limits of the system.

- A system boundary defines the scope of what a system will be. A system cannot have infinite use cases.
- The system boundary is shown as a rectangle spanning all the use cases in the system.
- System boundary of a system encloses the use cases of the system inside a rectangle and actors of the system outside the rectangle.

Example : In the same above statement about library, there is an association between actor 'student' and use-case 'borrow book'.

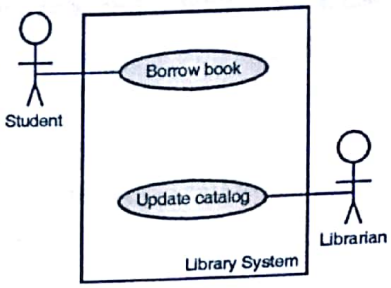


Fig. 4.3.5 : A Use case diagram showing the system boundary of a Library System

- But it is not that the system boundary represents the entire system always as in the above case - For large and complex systems, each of the modules may be enclosed in different system boundaries. And then the entire system can span all these modules finally into one system boundary.

Example :

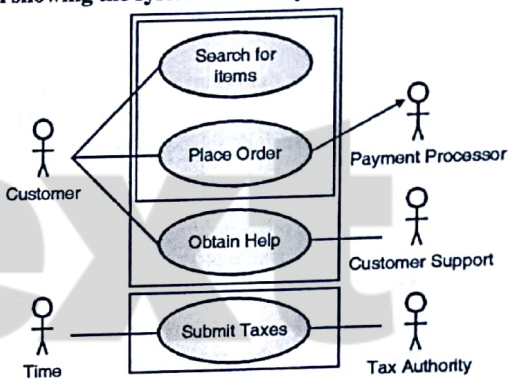


Fig. 4.3.6 : Example of System boundary boxes in use case diagram

- In the Fig. 4.3.6, Time is an external effect outside of the control of the system. When a certain time is reached that event is detected by the real-time clock that is inside the system and the use case starts automatically. It would be too difficult to hardcode the system always to start at a particular time, so normally an actor (i.e. Time) through another use case would be able to turn the automation on and off and set the time.

Example : Use-case diagram for Courseware Management System

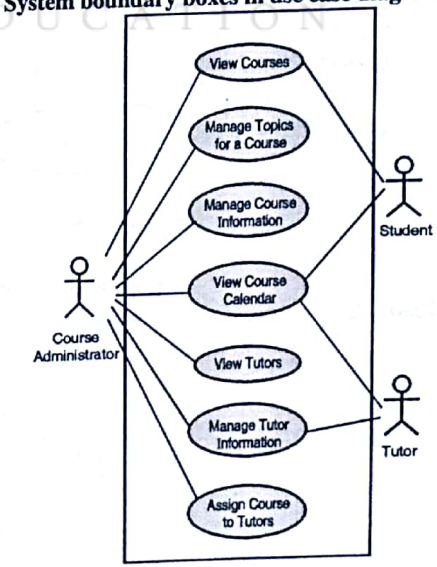


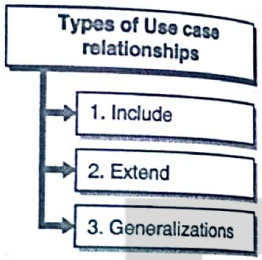
Fig. 4.3.7 : Use case diagram for Courseware Management System

4.3.3 Use Cases Relationships

- Use cases share different types of relationships. A relationship between two use cases is usually a dependency between those two use cases.
- Reusing the same use case in different types of relationships reduces the overall effort required in re-defining the use cases in a system.

Types of Use case relationships

Use-case diagrams support 3 types of relationships – *include, extend* and *generalization.*



1. Include

(Relationship from parent use case to inclusion use case)

- In this relationship, one use case references another use case.
- When one use case is using the functionality of another use case in a diagram, this relationship between the use cases is named as an include relationship.

Fig. C4.4 : Types of Use case relationships

- In an include relationship, a use case includes the functionality described in another use case as a part of its business process flow.
- An include relationship is depicted with a *directed arrow* having a *dotted shaft.*
  - The *base* of the arrowhead is the *parent use case.*
  - The *tip* of the arrowhead points to the *child use case.*
  - The *stereotype* "<<include>>" identifies the relationship as an include relationship.

Example : In library management system, the use case "Check for reservation" is contained within the "Borrow book" use case i.e. whenever the "Borrow book" use case executes, the scenario (business steps) defined in the "Check for reservation" use case is also executed.

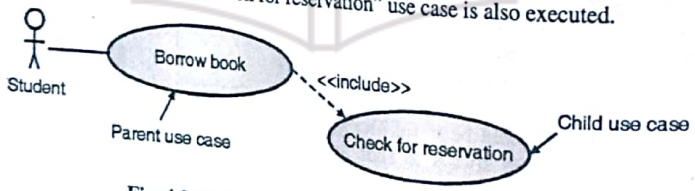


Fig. 4.3.8 : Example of <<include>> relationship

Example : Use case diagram for calculating in a Calculator depicting 'include' relationship. (Input is of the form "variable = expression")

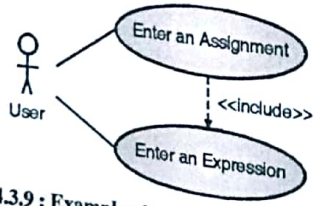


Fig. 4.3.9 : Example of <<include>> relationship

Example : Use case diagram for a Library Management System depicting 'include' relationship.

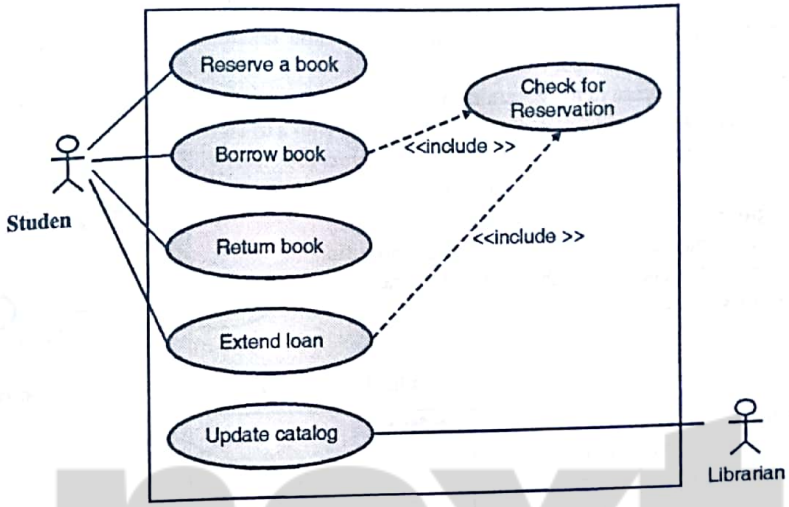


Fig. 4.3.10 : Use case diagram depicting 'include' relationship

2. Extend

(Relationship from extension use case to parent use case)

- Adds new behaviours or actions to a use case .
- This is used to add a use case that is similar to another use case but does a bit more or is more specialized.
- It expands the common behavior to fit the special circumstances.
- In an extend relationship, the child use case (extension use case) adds to the existing functionality and characteristics of the parent use case.
- An extend relationship is depicted with a directed arrow having a dotted shaft. (Extend relationship shows pointing of arrow just opposite to include relationship)
  - The *tip* of the arrowhead points to the *parent use case.*
  - The *base* of the arrowhead is the *child use case.*
  - The *stereotype* "<<extend>>" identifies the relationship as an extend relationship.

Example : The *extend* relationship between the "Distribute Fee Schedule" (parent) and "Distribute information to students" (extended) use cases. The "Distribute information to students" use case adds to the functionality of the "Distribute Fee Schedule" use case. The "Distribute information to students" use case is a specialized version of the generic "Distribute Fee Schedule" use case.

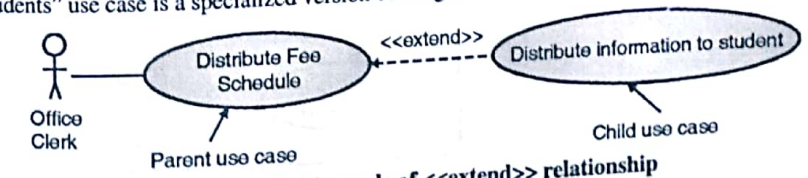


Fig. 4.3.11 : Example of <<extend>> relationship

3. Generalizations

(Relationship from a general use case to more specific use case)

- A generalization relationship is also a parent-child relationship between use cases like the above two relationships.
- The child use case in the generalization relationship is an enhancement of the parent use case.
- Generalization is depicted with a directed arrow with a triangle arrowhead from child to parent.
  - The *tip* of the arrowhead points to the *parent use case.*
  - The *base* of the arrowhead is the *child use case.*

Example : The *generalization* relationship between "transaction" (parent or more specific usecase) and "deposit", "withdrawal" and "transfer" (generalized) use cases.

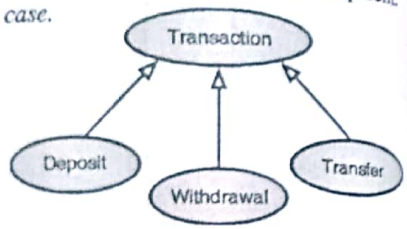


Fig. 4.3.12 : Example of a generalization relationship

Example : Use case diagram of ATM System depicting all the three relationships –include, extend and generalization.

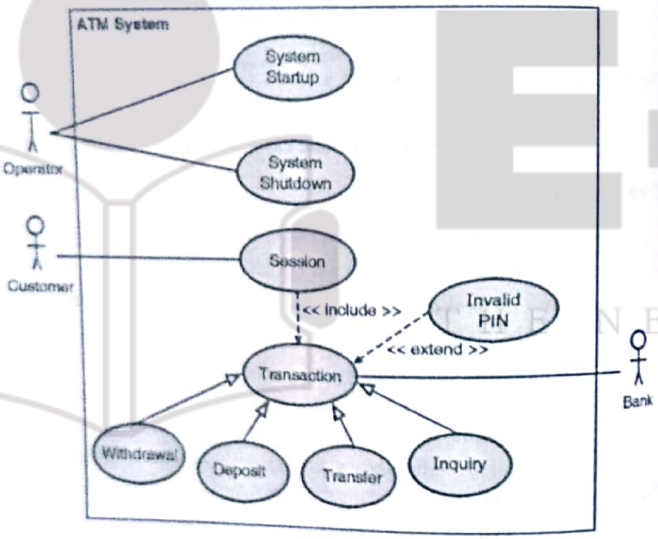


Fig. 4.3.13

Example : Use case diagram of Courseware Mgmt. System depicting all the three relationships – include, extend and generalization.

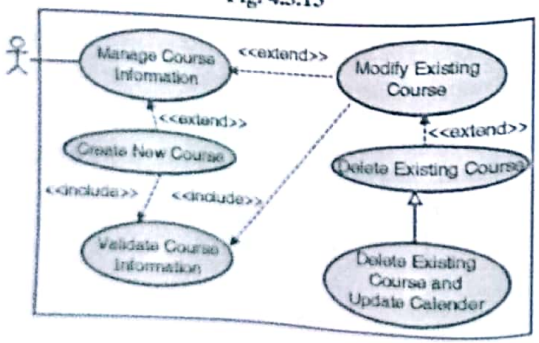


Fig. 4.3.14

Example : Use case diagram of Purchase Order System depicting all the three relationships – *include, extend and generalization.*

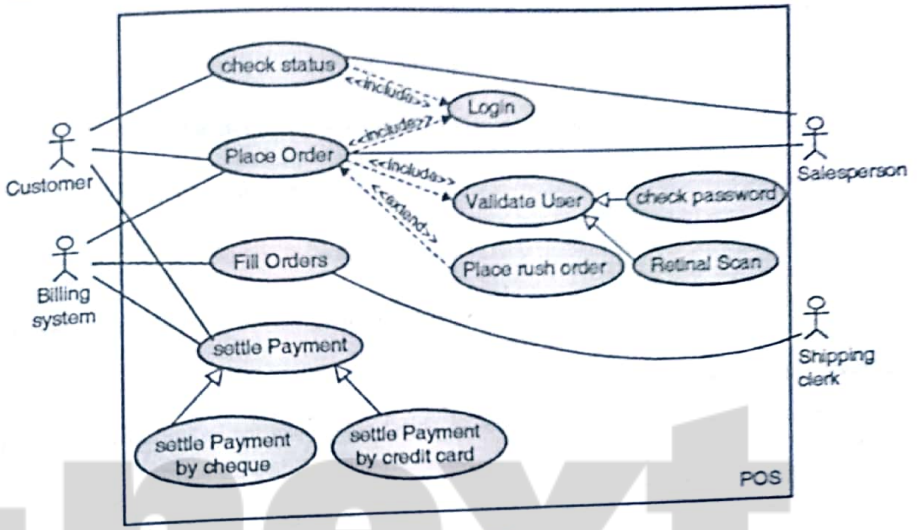


Fig. 4.3.15

Example : Use case diagram for Medical Clinic depicting all the three relationships –include, extend and generalization.

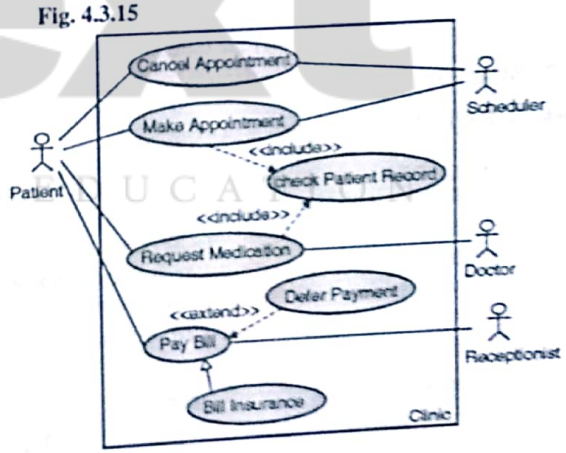


Fig. 4.3.16

Syllabus Topic : Class Diagram

4.4 Class Diagram

- This type of model includes classes, objects, attributes, operations and packages. We will see how to identify and design all these things.
- Here we will see the brief overview and various examples of class diagrams.

4.4.1 Identifying Classes and Objects

First, examine the problem statement (use cases) and then you can determine the classes by identifying each noun and entering it into a simple table. Classes mark themselves in one of the following ways:

- External entities (other systems, devices, and people) that produce or consume information to be used by a computer based system.
- Things (reports, displays, letters, signals) that are part of the information domain for the problem.
- Occurrences or Events that occur within the context of system operation.
- Roles (manager, engineer or salesperson) played by people who interact with the system.
- Organizational units (division, group and team) that is relevant to an application.
- Places (manufacturing floor or loading dock) that establish the context of the problem.
- Structures (Sensors, four wheeled vehicles or computers) that define a class of objects.
- To draw any type of UML diagrams, first we have to identify the classes and objects. And this is possible by studying the classification theory.

Classification Theory

- Classification is about identifying a class of an object rather than the individual objects within a system.
- Classification is the process of checking if an object belongs to a category or a class.
- Classes are important mechanism for classifying objects. The chief role of a class is to define the attributes, methods and its instances (objects).
- Classes are important as they are the conceptual building blocks for designing systems.

Strategies (Approaches) for identifying objects (Classes)

4.4.1(A) Noun Phrase Approach

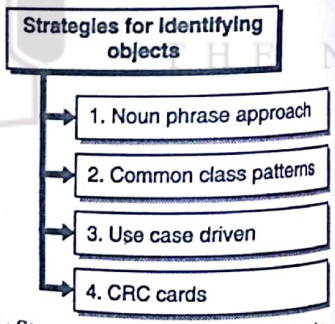


Fig. C4.5 : Strategies for identifying objects

- This approach was proposed by Rebecca Wirfs-Brock, Brian Wilkerson and Lauran Wiener.
- According to this approach, *Nouns are considered as candidate classes and verbs as its methods.* Nouns are converted into singular if they are in plural form.
- Nouns (Candidate classes) are divided into 3 categories :
  1. Relevant classes
  2. Irrelevant (can be skipped) classes
  3. Fuzzy classes
- Irrelevant classes do not have any purpose. They are unnecessary, so it is safe to slip it.
- Candidate classes are selected only from other 2 categories either by adding or removing some classes as necessary.

Guidelines for selecting 'tentative classes' in an application

- Circle or underline the nouns and noun phrases that occur in the requirements document(s); these become candidate classes (objects).

Example

- The system will keep track of membership information
- The system will manage inventory
- The system will facilitate the selling of bicycles

- All classes must make sense in the application domain; avoid computer implementation classes – place them to the design stage.

Guidelines for selecting 'candidate classes' from Relevant and Fuzzy Classes

You can challenge the initially selected candidate classes based on the following class rules.

- **Redundant classes :** Two classes cannot have same information. If more than one word is used to define the same data, select the one that is more meaningful in context of the system.
- **Adjective classes :** An adjective suggests a different kind of object, different use of same object or it could be utterly irrelevant. If the use of adjective signals that the behaviors of the object is different then make new class.

  Example : Adult members behave differently than Teenage members, so the two should be classified as different classes.
- **Attribute classes :** Tentative objects that are used only as values should be defined or restated as attributes and not as class.

  Example : Client Status and Demographic of Client are not classes but attributes of the Client class.
- **Irrelevant classes :** Each class must have a purpose and every class should be clearly defined and necessary. You must formulate a statement of purpose. Simply eliminate the candidate class.

The process of identifying relevant classes and eliminating irrelevant classes is an incremental approach. You can move back and forth among these steps as often as you like. This can be shown as in Fig. 4.4.1

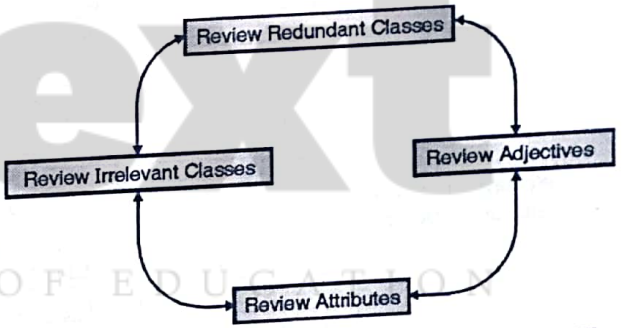


Fig. 4.19 : Moving back and forth among these classes (steps) as often as you like

Disadvantages

- Depends on the completeness and correctness of the requirements document which is rarely possible.
- Large volumes of text on system documentation might lead to too many candidate classes.

Example : Easy Money bank ATM system's requirements.

- The bank client must be able to deposit an amount to and withdraw an account from his or her accounts using the touch screen at the Easy Money ATM Machine. Each transaction must be recorded, and the client must be able to review all transactions performed against a given account. Recorded transactions must include the date, time, transaction type, amount and account balance after the transaction.
- An Easy Money bank client can have two types of accounts : checking account and savings account. For each account, one related savings account can exist.
- Access to the Easy Money bank accounts is provided by ATM Card having PIN code consisting of four integers between 0 and 9.

- One PIN code allows access to all accounts held by a bank client.
- No receipts will be provided for any account transactions.
- The bank application operates for a single banking institution only.
- Neither a checking nor a savings account can have a negative balance. The system should automatically withdraw cash from a related savings account if the requested withdrawal amount on the checking account is less than its current balance. If the balance on a savings account is less than the withdrawal amount requested, the transaction will stop and the bank client will be notified with a message.

Solution : Initial study produces the following noun phrases :

Account	Client's Account
Account Balance	Four Digits
Amount	Message
ATM Card	Password
ATM Machine	PIN
Bank	PIN Code
Bank Client	Receipts
Card	Record
Cash	Savings
Checking	Savings Account
Checking Account	System
Client	Transaction

We have to select candidate classes from relevant and fuzzy class category, so it's safe to eliminate the irrelevant classes. Strike out the eliminated classes.

Account	Client's Account
Account Balance	Four Digits (irrelevant class)
Amount	Message
ATM Card	Password
ATM Machine	PIN
Bank	PIN Code
Bank Client	Receipts
Card	Record
Cash	Savings
Checking	Savings Account
Checking Account	System
Client	Transaction

Now, according to the redundant class guideline, eliminate the classes that describe the same idea. We have to select class i.e. most meaningful. Thus, reviewed list of candidate classes is :

Account	Client's Account (Account, Client's Account = Account)
Account Balance	Four Digits (irrelevant class)
Amount	Message

Account	Client's Account (Account, Client's Account = Account)
ATM Card	Password
ATM Machine	PIN (PIN, PIN Code = PIN Code)
Bank	PIN Code
Bank Client	Receipts
Card (ATM Card, Card = ATM Card)	Record
Cash	Savings (Savings, Savings Account = Savings Account)
Checking (Checking, Checking Account = Checking Account)	Savings Account
Checking Account	System
Client (Client, Bank Client = Bank Client)	Transaction

Now, according to the adjective class guideline, create new classes if the classes behave differently with their adjective. But, in this example, there are no adjectives with any of the classes. Thus, there is no change. Now, according to the attribute class guideline, eliminate the nouns that are attributes, not classes.

Account	Client's Account (Account, Client's Account = Account)
Account Balance (An attribute of the Account class)	Four Digits (irrelevant class)
Amount (A value, not a class)	Message (A value, not a class)
ATM Card	Password (An attribute of the Bank Client class)
ATM Machine	PIN (PIN, PIN Code = PIN Code)
Bank	PIN Code (An attribute of the Bank Client class)
Bank Client	Receipts
Card (ATM Card, Card = ATM Card)	Record
Cash	Savings (Savings, Savings Account = Savings Account)
Checking (Checking, Checking Account=Checking Account)	Savings Account
Checking Account	System
Client (Client, Bank Client = Bank Client)	Transaction

Now, reviewing the class purpose, each class must have a purpose; else eliminate that class which adds no purpose to the system. Cash and Receipts do not add purpose to the system. The final candidate classes are :

Account
(It is an abstract class that defines the common behaviors that can be inherited by checking or savings account)
ATM Card
(Provides a client with a key to access his account)
ATM Machine
(Provides interface to EasyMoney Bank)
Bank
(Bank clients belong to the bank)
Bank Client
(A client is an individual who has an account in the bank)
Checking Account
(Provides more specialized withdrawal service to the client's account)
Savings Account
(Models a client's savings account)
Transaction
(Keeps track of transaction, time, date, type, amount and balance)

4.4.1(B) Common Class Pattern Approach

The common class patterns approach is based on knowledge of common classes that has been proposed by researchers.

The patterns used for finding the candidate class and object using this approach are :

Pattern Name	Description	Example
Concept class	Principles or ideas. Are non-tangible	Performance
Events class	Things that happen at a given date and time, or as steps in an ordered sequence. These are associated with attributes such as : who, what, when, where, how or why.	Landing, Request, Interrupt, Order
Organization class	Formally organized collections of people, resources or facilities having a defined mission.	Departments
People class	Different roles that users play in interacting with the application. Two categories of people class : 1) People who use the system. 2) People who do not use the system but about whom information is kept by the system.	Teachers, Students, Employees
Places class	Areas set aside for people or things. Physical locations that the system must keep information about.	Travel Office, Buildings
Tangible things and devices class	Physical objects that are tangible and devices with which the application interacts	Cars, Sensors

Disadvantages

- Loosely bound to user requirements.

- Naming misinterpretations are possible.

Example : Take the same above Easy Money Bank ATM system. We can classify the nouns based on common class pattern approach as follows :

Pattern Name	Classes
Concept class	Not Applicable
Events class	Account, Checking Account, Savings Account, Transaction
Organization class	Bank
People class	Bank Client
Places class	Not Applicable
Tangible things and devices class	ATM Machine

4.4.1(C) Use Case Driven Approach

- Use-case modeling is a problem-driven (function-driven) approach in which the designer first considers the problem (functions or say use-cases) rather than considering the relationship between objects.
- This approach helps to understand the behavior of the system's objects.
- Use cases are used to model the scenarios in the system and specify which external actors interact with the scenarios. The scenarios are described through sequence of steps.
- The designer then examines the steps of each scenario to find out what objects are needed for the scenario to occur.
- At least one scenario must be prepared for each different use-case instance. Each scenario shows different sequence of interaction between actors and the system.
- Designer walks through each scenario to identify the objects, responsibilities of each object and how these objects interact with each other.

Example : Consider the same above case study of Easy Money Bank.

Assume a use case scenario, say 'Invalid PIN'

Use-case name : Invalid PIN

Step 1 : Insert ATM Card
Step 2 : Request PIN
Step 3 : Insert PIN
Step 4 : Verify PIN
Step 5 : If invalid PIN, display bad message
Step 6 : Eject ATM Card

Based on these activities, we find that the classes that interact with each other are : Bank Client, ATM Machine and Bank Operator.

Disadvantage

Relies on the completeness of use case scenarios.

4.4.1(D) CRC Approach

- A Class Responsibility Collaborator (CRC) model was developed by Beck and Cunningham in 1989, Wilkinson in 1995, Ambler in 1995.

- CRC is an effective way to identify classes based on the analysis of how objects collaborate to perform business functions (use cases).
- CRC Card Modeling is a simple but powerful object-oriented hands-on analysis technique.
- It was first developed as a teaching tool, then used as a collaborative technique to involve users, analysts and programmers in the design process
- It is a collection of standard index cards that is divided into 3 sections : *class name, responsibilities and collaborators.*

Class Name	
Responsibilities (attributes and operations)	Collaborators (relationships)

Fig. 4.4.2 : Three sections of CRC (4" × 6" card)

1. A ***class name*** represents a collection of *similar objects*. It appears in upper left hand corner.

Example : In a university system, classes would represent students, professors, and seminars. The name of the class appears across the top of a CRC card and is typically a singular noun or singular noun phrase, such as *Student*, *Professor*, and *Seminar*.

2. A ***responsibility*** is something that a class *knows* or *does*.

Example : Students have names, addresses, and phone numbers. These are the things a student *knows*. Students also enroll in seminars, drop seminars, and request transcripts. These are the things a student *does*. The things a class knows and does constitute its responsibilities. Also, a class can change values of the things it knows, but it is unable to change the values of what other classes know.

3. A ***collaborator*** is another class that a class interacts with to fulfil its responsibilities. Sometimes a class has a responsibility to fulfil, but not have enough information to do it. For example, students enrol in seminars. To do this, a student needs to know if a spot is available in the seminar and, if so, he then needs to be added to the seminar. However, students only have information about themselves (their names and so forth), and not about seminars. What the student needs to do is collaborate/interact with the card labelled *Seminar* to sign up for a seminar. Therefore, *Seminar* is included in the list of collaborators of *Student*.

Student	
Name Address Phone Number Enroll in Seminar Drop a Seminar Request transcripts	Seminar

Fig: 4.4.3 : Student CRC card

Creating CRC Models

The process of creating CRC model consists of three steps :

Step 1: Identify classes' responsibilities (and identify classes)

- Classes are identified and grouped by common attributes, which also provide candidates for superclass.
- Once you have identified potential classes (e.g. by finding nouns in the requirements specification), you use one card for each class.
- The class names are written on the CRC cards. The card also notes sub and superclass to show the class structure.

Step 2: Assign responsibilities

- On this card, you write down the 'responsibilities' of that class. These are the potential methods.
- Responsibilities are distributed. They should be as general as possible and placed as high as possible in the inheritance hierarchy.

Step 3: Identify collaborators

- Once you agree that the classes & responsibilities are OK, you find out for each class, which other classes it need to collaborate with and write this on the card.

  Collaboration is done either 1) to request for information or 2) to request to perform a task

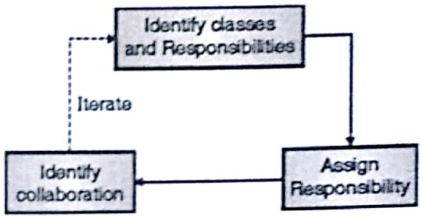


Fig. 4.4.4: CRC Process

Move the Cards around

To help everyone to understand the system, the cards must be placed on the table in an intelligent manner so that two cards that collaborate with each other must be placed close together, whereas two cards that don't collaborate must be placed far to each other.

In this manner, the more two cards collaborate; the closer they should be on the table. By placing the cards that collaborate with one another close together, it is easier to understand the relationships between classes.

customer	
Name Address Phone Number Place order Knows order History	order

order	
Order Date Delivery Date Order Number Order Items Make Total Applicable Taxes	Items

Fig. 4.4.5: CRC Cards of 'Purchase Order'

Example : Few CRC Cards of Purchase Order.

Example : CRC cards for University Enrollment.

Student	
Student Name Address Phone Number Enroll in Degree	Degree Course

Course	
Course Code Course Name	Degree

Degree	
Degree Code Degree Name	Study Program

StudyProgram	
Years Semesters	

Fig. 4.4.6 : CRC Cards for 'University Enrollment'

Example : CRC cards for Seminar Enrollment.

Student	
Student Name Address Phone Number Email-Id Marks Received Validating Information	Enrollment

Enrollment	
Student Name marks Grade Seminar Topic	Seminar

Seminar	
Name Seminar Number Fees Waiting list Enrolled Student Instructor Add Student Drop Student	Student Professor

Professor	
Professor Name Address Phone number Email-Id Salary Provide Information Instructing seminars	Seminar

Fig. 4.4.7 : CRC Cards for 'Seminar Enrollment'

We need to identify the responsibilities because

- Responsibilities identify problems that are to be solved.
- A responsibility serves as a handle for discussing potential solutions.
- Once the system's responsibilities are understood, we can start identifying the attributes of the system's classes. Note that, responsibilities include *attributes* and *methods* of the class.

4.4.2 Identifying Attributes

Attributes correspond to nouns followed by preposition phrases or to adjectives or adverbs.

- Omit derived attributes - They should be expressed as a method.

Example : In the same Easy Money ATM system example, we will state the attributes of Account, ATM Machine, Bank Client and Transaction classes.

Account	ATMMachine	BankClient	Transaction
number balance	address state	firstname lastname pinnumber cardnumber accountnumber	transID transDate transTime transType amount post Balance

Fig. 4.4.8

4.4.3 Defining Methods

- Methods are defined as services that the objects must provide.
- Methods are the events that occur between objects of the class.
- An event is considered to be an action that transmits information; therefore these are the operations that the objects must perform.
- Methods also can be derived from the use case scenario testing.

Example : In the same EasyMoney ATM system example, we will state the methods of Account, ATM Machine, BankClient and Transaction classes.

Account	ATMMachine	BankClient	Transaction
number balance deposit() withdraw() createTransaction()	address state	fristname lastname pinnumber cardnumber verifyPassword()	transID transDate transTime transType amount postBalance

Fig. 4.4.9

4.4.4 Finalizing the Object (Class) Definition

Table 4.4.1 : Various ways of naming a Class

Description	Example
Use a noun or noun phrase	my latest books
Should be singular and not plural	my latest book
Avoids possessives	a latest book
Doesn't contain irrelevant adjectives	a book
Uses initial capital letters	A Book
Doesn't have spaces between words	ABook
Doesn't contain articles (a, an, the) pronouns	Book

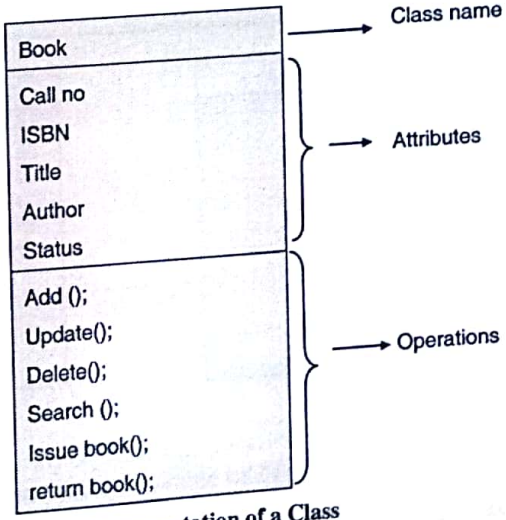


Fig. 4.4.10 : Representation of a Class

Example: Class Diagram of Library Management System

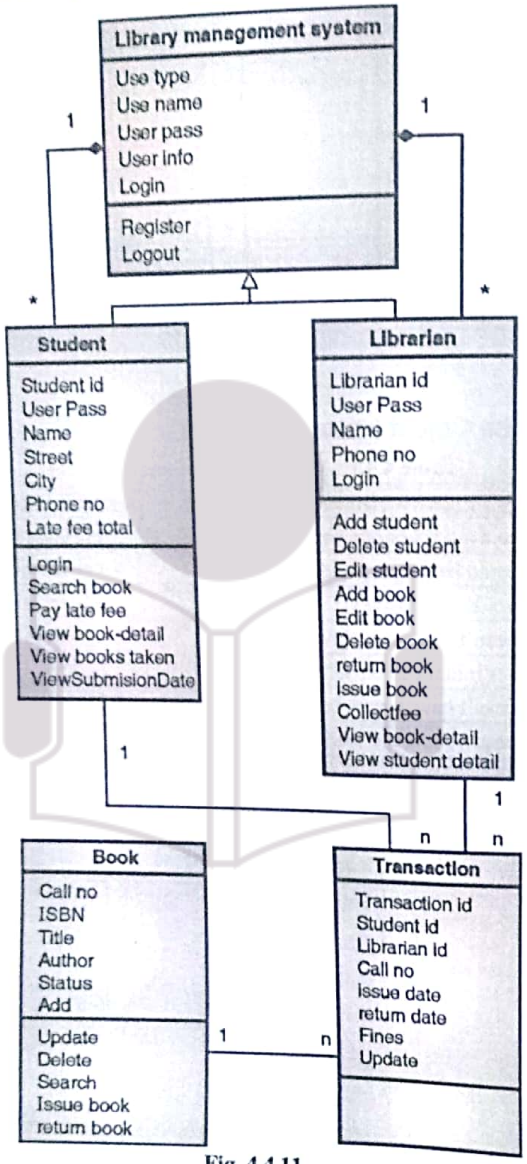


Fig. 4.4.11

Example : Draw class diagram for Private course management system where it has various courses. Each course consists of various topics. Course has course calendar and It is managed by course administrator. Course administrator can manage some tutors for each course. Many students can be admitted in each course.

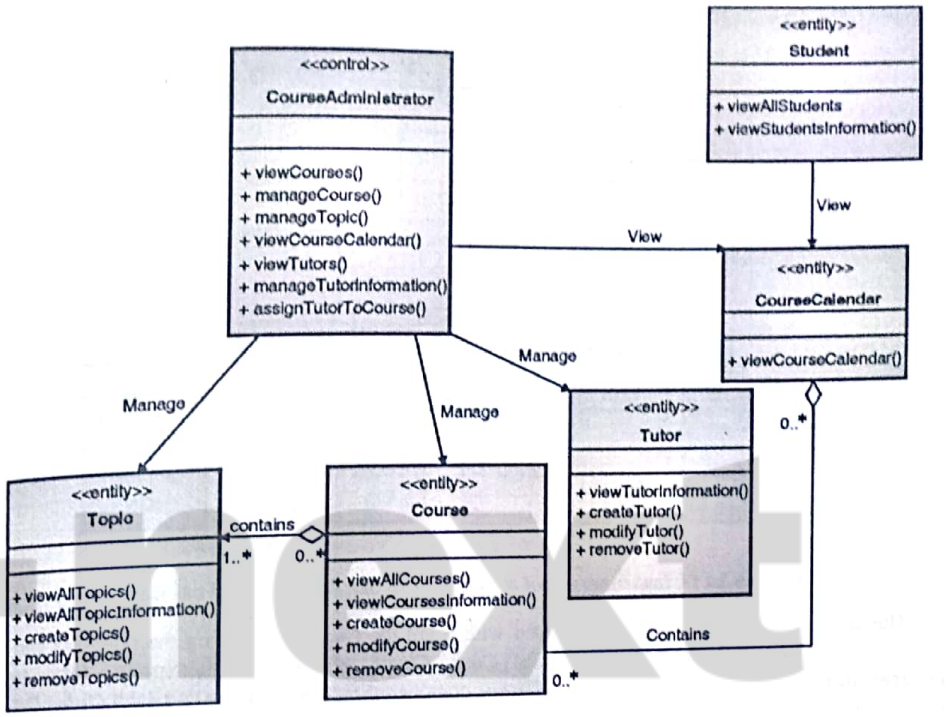


Fig. 4.4.12

Example: Draw class diagram for online shopping.

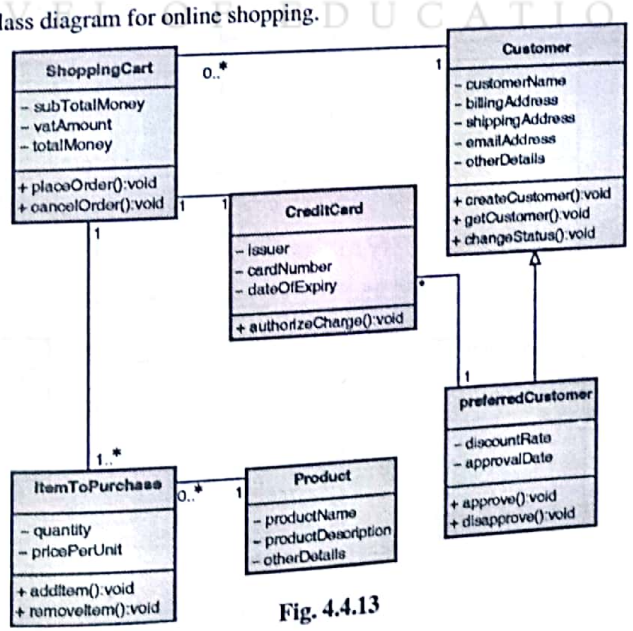


Fig. 4.4.13

Example : The class diagram of a customer order from a retail catalog.

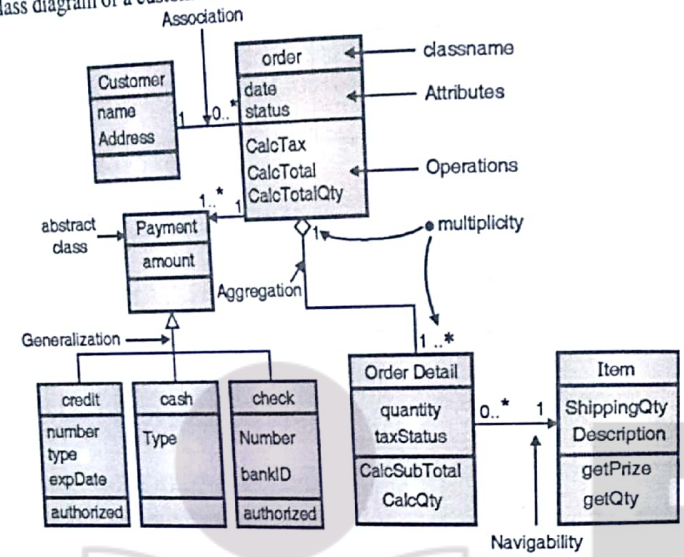


Fig. 4.4.14 : Class diagram of a customer order from a retail catalog

The central class is the *Order*. **Associated** with it are the *Customer* making the purchase and the *Payment*. A *Payment* is one of three kinds: *Cash*, *Check*, or *Credit* demonstrating **inheritance (generalization)**. The *Order* contains *Order Details* demonstrating **aggregation** and each *Order Details* is associated with *Item*.

Example : Draw class diagram for Seminars given by students from different courses.

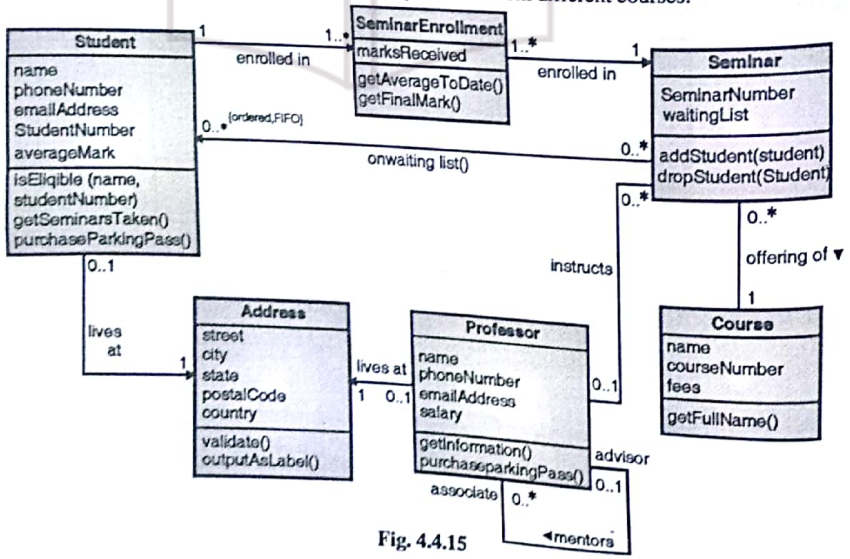


Fig. 4.4.15

Example : Class diagram for Hospital management where it has various departments. If each department consists of various doctors out of which, one of the doctors is the head of the department. Patient takes treatment from Doctors where treatments are based on Laboratory Result. Patient can be an Indoor Patient or Outdoor Patient. Hospital has various operation theatres used for various operations by various doctors on indoor patient.

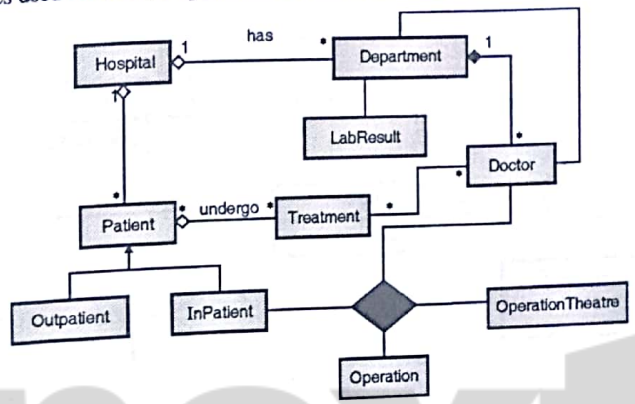


Fig. 4.4.16

Example: Class diagram for word processor.

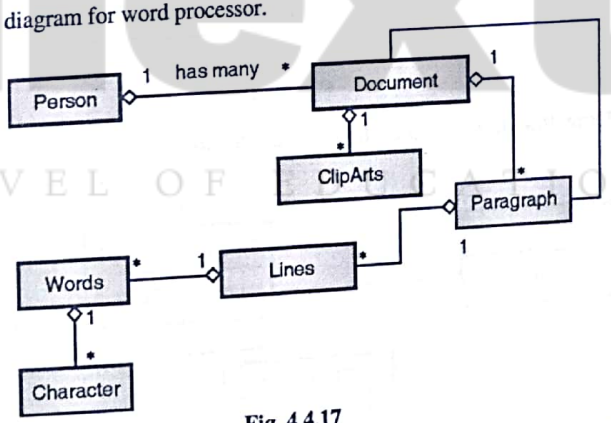


Fig. 4.4.17

Syllabus Topic : Object Diagram

4.5 Object Diagram

Here we will see just the various examples.

Example : Draw the object diagram for Hospital management system.

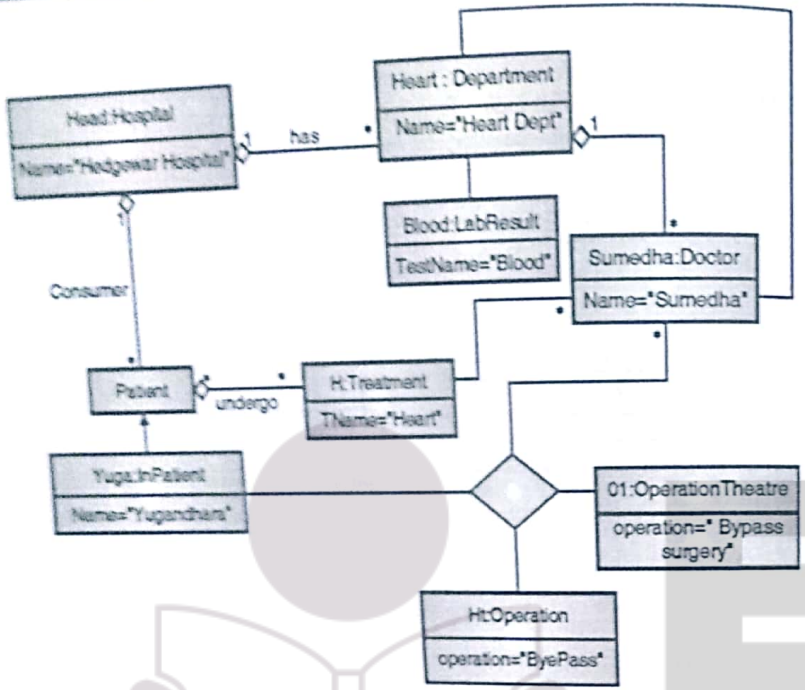


Fig. 4.5.1

Example : Draw the object diagram for word processor.

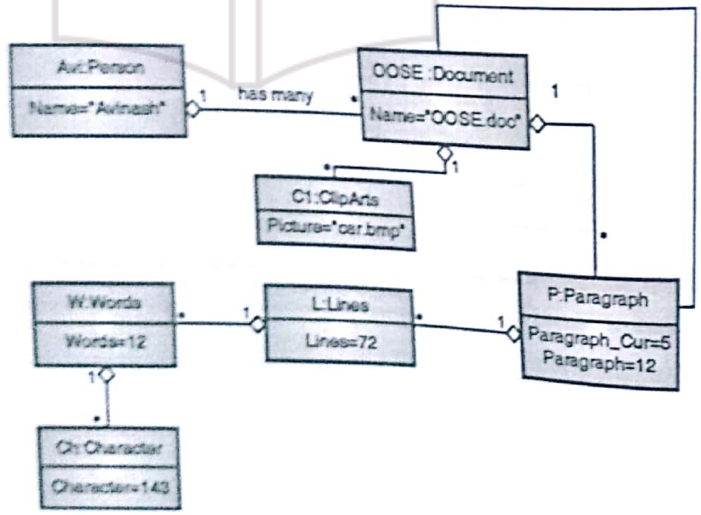


Fig. 4.5.2

Example : Draw the Object diagram for car rental system.

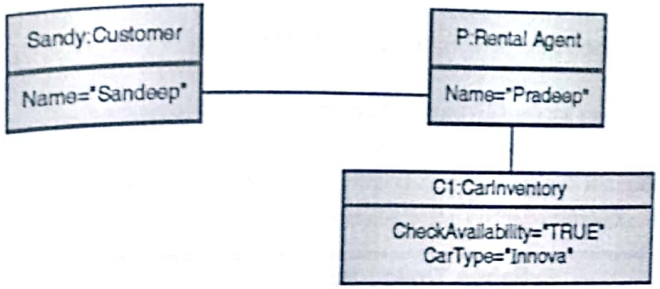


Fig. 4.5.3

4.6 Packages

- You draw a class model on a single page for many small and medium sized problems. But it is very difficult to understand the entirely large model, thus, it is better to partition the larger class models so that it is easily understood. These partitions can then be put into packages.
- A package is a group of elements (classes, associations, generalizations and smaller packages) with a common theme. Packages are given a representative name.
- Packages form a tree with increasing abstraction towards the root, which is the top level package or the entire application.

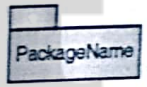


Fig. 4.6.1 : Notation for a package

Example : Package diagram for Seminar enrolment by students from different courses.

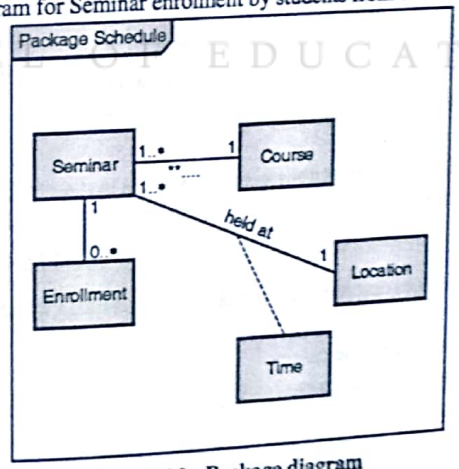


Fig. 4.6.2 : Package diagram

4.7 Interaction Diagram

- Describes the collaboration between groups of objects. The collaboration is required to get the job done.

- Used to model the behavior of several objects in a use case. This is possible using a Sequence diagram.
- Demonstrates collaboration between the different object. This is possible using a Collaboration diagram.
- The communications are partially ordered based on time or in a sequence.

UML supports two kinds of Interaction diagrams:

1. **Sequence Diagram** displays the time sequence of the objects participating in the interaction.
2. **Collaboration Diagram** displays interaction around the objects and their links to one another.

Syllabus Topic : Sequence Diagram

4.7.1 Sequence Diagram

- They represent how objects interact with each other to perform the behaviours as stated in a use case scenario.
- A sequence diagram shows a series of messages exchanged by a selected set of objects with emphasis on the chronological course of events.
- The focus is less on messages themselves and more on the **order** in which messages are exchanged.
- Sequence diagrams communicate what messages are sent between a system's objects as well as the order in which they occur.
- Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario.
- UML sequence diagram is a dynamic modelling technique because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.
- Is used primarily to show the interactions between objects in the sequential order of their occurrence.

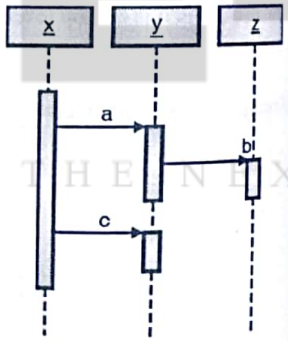


Fig. 4.7.1 : Sequence diagram

- They are used to validate and describe the logic of a ***usage scenario*** which may be :
  - part of a use case or
  - an entire pass through a use case or
  - a pass through several use cases.

Scenario

- A scenario is a sequence of events that occur during one particular execution (use case) of a system.
- A scenario can include all events in the system or can only include only those events that are generated by a certain object in that system.
- A scenario is usually structured with numbered steps in a sequence and calls to sub-sequences.

Example : Use case Scenario for phone call

1. Caller lifts receiver
   - 1.1 Dial tone begins
   - 1.2 Caller dials a digit
   - 1.3 Dial tone ends
   - 1.4 Caller dials remaining digits
2. Ringing tone starts at caller side and receiver also hears the phone ring
3. Receiver answers the phone
4. Ringing stops on caller and receiver side
   - 4.1 Phone is connected. Caller and Receiver talk to each other
5. Receiver hangs up.
6. Phone connection is broken on both sides.

Designing a sequence diagram

Two dimensions of Sequence diagram :

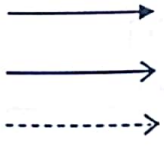
1. The *vertical dimension* (top to bottom) of the sequence diagram shows the ***time*** sequence of messages as they occur.
2. The *horizontal dimension* shows the ***object*** instances that the messages are sent to.

Diagram Elements in a Sequence diagram

- **Actor :** Represents an external person or entity that interacts with the system.

user

- **Object :** Represents an object in the system or one of its components.
- **Lifelines :** Each vertical *dotted line* is a lifeline representing the time limit that an object exists. A lifeline has a rectangle containing its object name. If its name is caller then the lifeline represents the classifier which owns the sequence diagram.
- **Messages:** Messages are displayed as arrows. Messages can be complete, lost or found; synchronous or asynchronous; call or signal. The synchronous message is denoted by the *solid* synchronous or asynchronous; call or signal. The synchronous message is denoted by the *solid arrowhead*, asynchronous message *by line arrowhead*, and the return message by *dashed line*. Each message is labeled with a message name.



Synchronous Message

Asynchronous Message

Return Message

- **Activation bar :** A message (an arrow) goes from the sender to the top of the *activation bar* of the receiver's lifeline. A thin rectangle running down the lifeline denotes the *execution occurrence*, or *activation bar* of a focus of control. The *activation bar* represents the duration of execution of the message.

Example : Sequence diagram for a Phone Call.

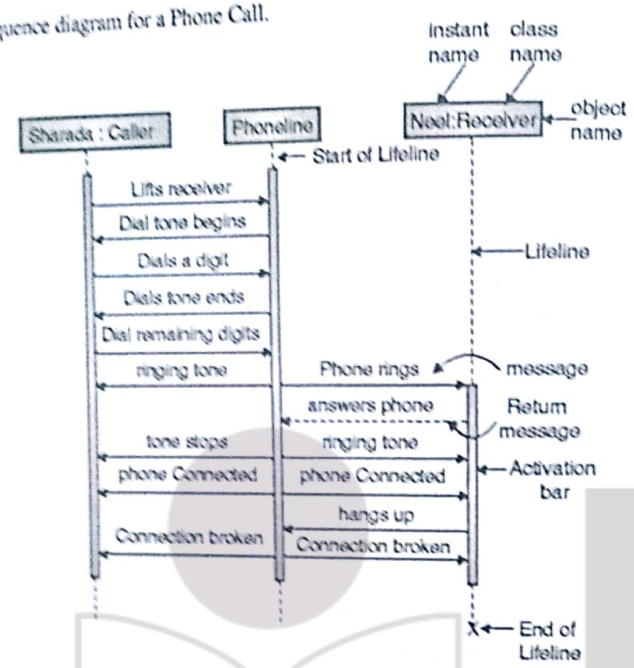


Fig. 4.7.2 : Sequence diagram for a Phone Call

Example : Simple Sequence diagram of Courseware Management System.

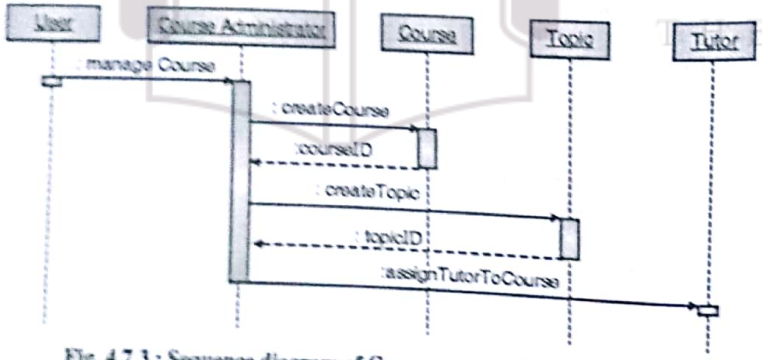


Fig. 4.7.3 : Sequence diagram of Courseware Management System

- **Self call** (Self-delegation) **:** It is a message that object sends to itself.
- **Lifeline Start and End :** A lifeline may be *created* or *destroyed* during the timescale represented by a sequence diagram.
  - To destroy, the lifeline is terminated by a stop symbol, represented as a *cross*.
  - And to create, the symbol at the head of the lifeline is shown at a lower level down the page than the symbol of the object that caused the creation.

- **Part Decomposition :** An object can have more than one lifeline coming from it. This allows for *inter-object* and *intra-object* messages to be displayed on the same diagram.

Example : Sequence diagram showing 'Part Decomposition'.

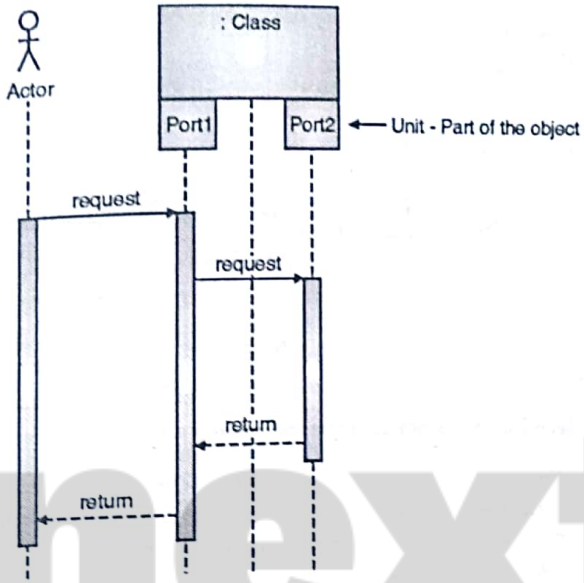


Fig. 4.7.4 : Sequence diagram showing 'Part Decomposition'

- **State Invariant:** A state invariant is a constraint placed on a lifeline that must be true at run-time. It is shown as a rectangle with semi-circular ends.

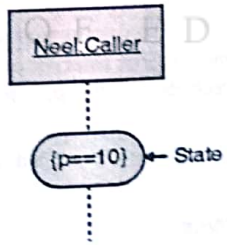


Fig. 4.7.5 : Showing state invariant

- **Condition:** The message is sent only if the condition is true.
  syntax: [expression] message-label
- **Iteration:** The message is sent many times to possibly multiple receiver objects.
  syntax: * [expression] message-label

Example : Sequence diagram for making a hotel reservation.

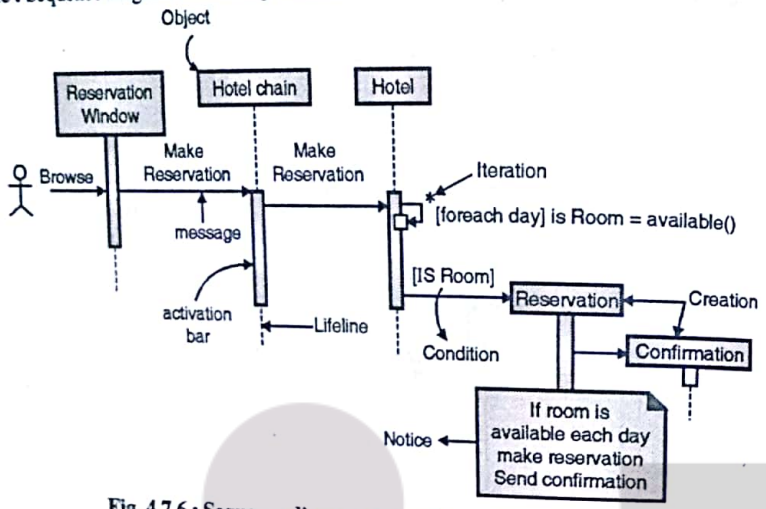


Fig. 4.7.6 : Sequence diagram for making a Hotel Reservation

- The object *(user interface)* of the class *Reservation window* initiates the messaging.
- The *Reservation window* sends a *make Reservation()* message to a *Hotel Chain*. The *Hotel Chain* then sends a make Reservation() message to a *Hotel*. If the *Hotel* has available rooms, then it makes a *Reservation* and a *Confirmation*.
- Every vertical *dotted line* is called as the **lifeline** that describes the time duration that an object exists for. Each *arrow* is called as a **message call** and it goes from the sender to the top of the **activation bar** on the receiver's lifeline. The *activation bar* represents the time duration of message execution.
- In our diagram, the *Hotel* issues a **self call** to determine if a room is available. If so, then the *Hotel* creates a *Reservation* and a *Confirmation*. The **asterisk** on the self call means **iteration** (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, [ ], is a **condition**.
- The diagram has a clarifying **note**, which is text inside a dog-eared rectangle. *Notes can be put into any kind of UML diagram.*

Advantages of Sequence diagrams

1. **Helps you to discover architectural, interface and logic problems early in SDLC process**

   This is specially true for systems involving interaction of components that are being implemented in parallel by different teams. In the phone call example, each task is implemented by a separate team. Having a set of sequence diagrams describing how the interfaces are actually used and what messages/actions are expected at different times gives each team a consistent implementation plan.

2. **Collaboration tool**

   - Sequence diagrams are valuable collaboration tools during design meetings because they allow you to discuss the design in concrete terms - you can see the interactions between entities, various state transitions and alternate cases on paper.

3. **Sequence diagram editor**

   - It become easy to edit your sequence diagrams - you can make the corrections during the meeting itself and instantly see the result of the changes as you make them.

4. **Documentation**

   - Sequence diagrams can be used to document the dynamic view of the system design at various levels of abstraction which is often difficult with static diagrams.

Syllabus Topic : Collaboration Diagram

4.7.2 Collaboration Diagram

- While sequence diagrams focus on the *interactions of objects over time*; the collaboration diagrams focus on the *relationships between objects.*
- Collaboration defines the communication pattern performed by set of instances.
- Collaboration diagram is almost similar to sequence diagram but only difference is that of *viewpoint* - Sequence diagram cannot illustrate the relation between objects where as collaboration diagram does.
- The collaboration diagram models how the interactions utilize the structure of participating objects and their relationships. It represents the relationship and interaction between software objects.
- Collaboration diagram illustrates messages being sent between classes and objects (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

Use of Collaboration diagram

- Collaboration diagrams are used in the systems where structure is important to concentrate on the effects of the instances.
- This shows interaction between classes but focuses on structural relationships between objects.

Designing a Collaboration diagram

- Wherever a message is to be sent, there must be a *link* i.e. an association must exist between the classes. Also, the association must be navigable in the required direction.
- *Actors* can be shown as in a use case diagram.
- Each ***object*** is shown as rectangle, which is labelled with either class or object name or both. If both class name and object name are used, Class names are preceded by colons (:) i.e. *objectName: className* (which was underlined in earlier versions; not now days).

Example: 'cat is a pet' where, cat is object and pet is class.

- Objects are linked through message paths. ***Links*** between objects are shown same like associations in the class model.
- *Messages* are attached to the message paths. Messages have a sequence number to denote time of occurrence. Each message in a collaboration diagram has a *sequence number*. The top-level message is numbered 1. Messages at the same level (sent during the same call) can have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur such as 1.1, 1.2...But it is easy to read simple numbering in collaboration diagrams than the decimal numbering. In a collaboration diagram numbering the message indicates the sequence.

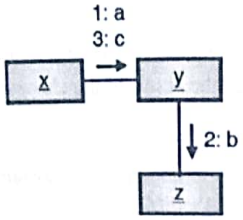


Fig. 4.7.7 : Collaboration diagram

Table 4.7.1 : Notations Used in Collaboration Diagram

Sr. No.	Notation	Use
1.		Actor :
2.	Alok: staff	Instance of class. Labeling as [instance name][: instance type]
3.	College:Staff	Multi Objects Labeling as [name][:type]
4.		Message Directions

Example : Collaboration diagram for making a Hotel Reservation.

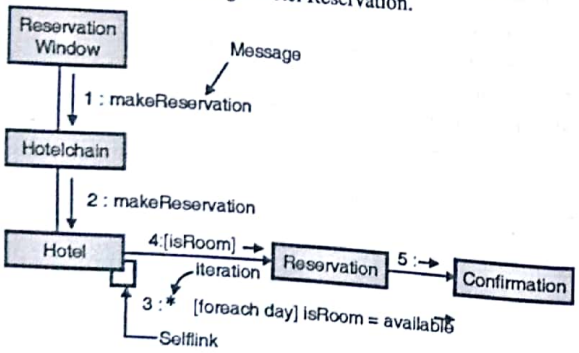


Fig. 4.7.8 : Collaboration diagram for making Hotel Reservation (simple numbering)

Example : Collaboration diagram for a Phone Call.

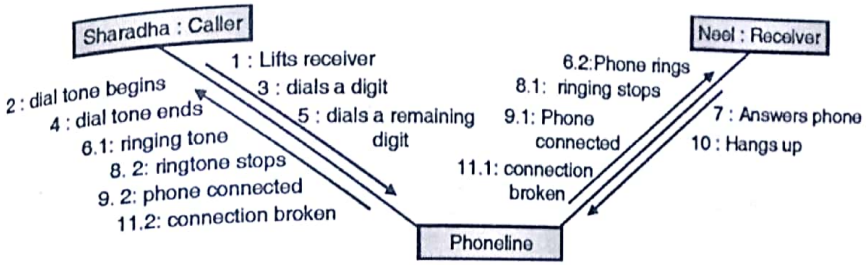


Fig. 4.7.9 : Collaboration diagram for a Phone Call (decimal numbering)

Example : Collaboration diagram for Courseware Management System.

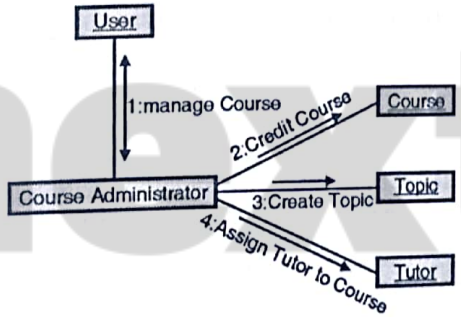


Fig. 4.7.10 : Collaboration diagram for Courseware Management System

Example : Collaboration diagram for Railway Reservation Systems :

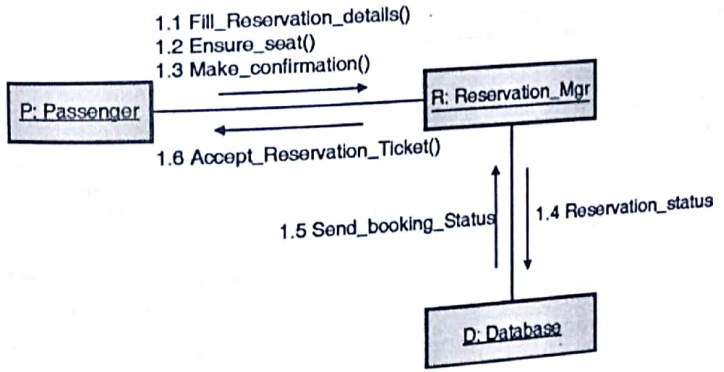


Fig. 4.7.11

Example : Collaboration diagram for E-Product Purchasing :

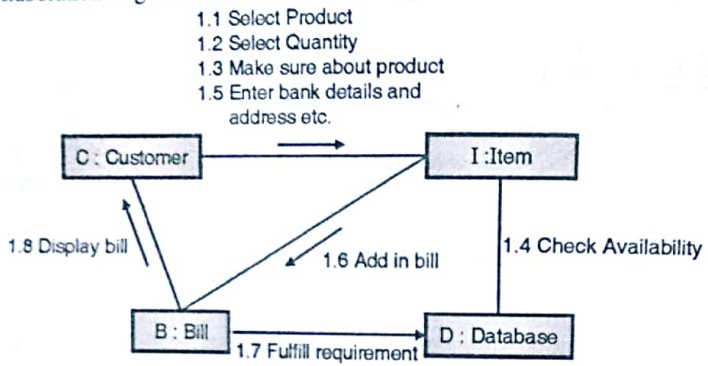


Fig. 4.7.12

Similarities and Differences in Sequence and Collaboration diagram

Similarities	1. Sequence & Collaboration diagrams together form the interaction diagrams.	
	2. These diagrams describe the same information, and can be transformed into one another.	
Differences	Sequence diagrams focus on the order in which the messages are sent.	Collaboration diagrams focus on relationships between objects.
	Sequence diagram is useful for describing the procedural flow through many objects, and for finding race conditions in concurrent systems.	Collaboration diagram is useful for visualising the way several objects collaborate to get a job done, and for comparing a dynamic model with a static model.

Syllabus Topic : State-chart Diagram

4.8 State-chart Diagram

- A **state** is a mode or condition of being.
- A state diagram is a dynamic model showing changes of state that an object goes through during its lifetime in response to events.
- State diagrams (also called State Chart diagrams) are used to help the developer better understand any complex functionality of specialized areas of the system.
- In short, State diagrams depict the dynamic behaviour of the entire system, or a sub-system, or even a single object in a system.

Designing a State Diagram

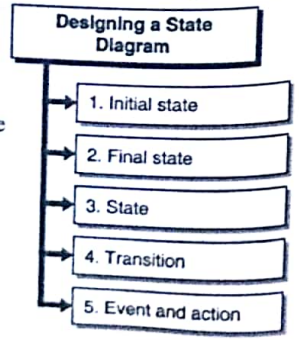


Fig. C4.6 : Designing a State Diagram

1. Initial state

The filled in circle shows the starting point or first activity of the diagram. This is also called as a **"pseudo state,"** where the state has no variables and no activities.

2. Final state

The filled circle with a border (bull's eye symbol) is the end of the state diagram. This is also a pseudo state because it does not have any variable or action described. A state diagram can have zero or more final states.

3. State

Represents the state of object at an instant of time. State is a recognizable situation and exists over an interval of time. This is denoted by a rounded rectangle and compartments within to describe state-name, variables, and activities.

state-name
variable
activities

4. Transition

A transition is a change from one state to other is indicated by an arrow. The *event* and *action* causing the transition are written beside the arrow, separated by a slash.

Event[Condition]/Action

- Transitions that occur when the state completed an activity are called **trigger less** transitions.
- If an event has to occur after the completion of some event or action, that event or action is called the guard condition (depicted by square brackets around the description of the event/action in the form of a Boolean expression). The transition then takes place after the guard condition occurs.

5. Event and action

A trigger that causes a transition to occur and changes the state is called as an event or action. An event occurs at a particular time has no duration. As described above, an event/action is written above a transition that it causes.

Example : State diagram for a phone call

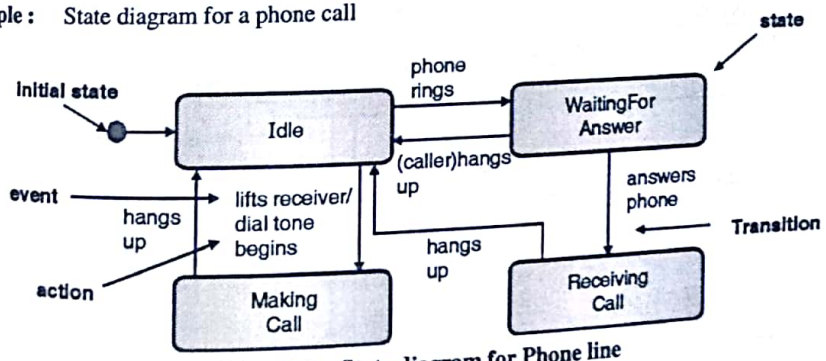


Fig. 4.8.1 : State diagram for Phone line

History states

A flow may require that the object go into wait state, and on the occurrence of a certain event, go back to the state it was in. This is shown with the help of a letter H enclosed within a circle.

Example : Showing History States.

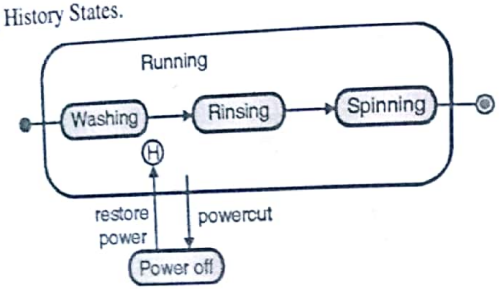
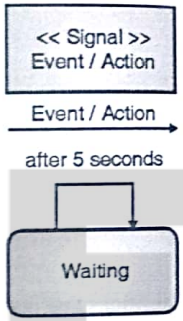


Fig. 4.8.2 : showing History States

- **Signal :** When an event produces a message or a trigger to be sent to a state that in-turn causes the state transition; then, that message is called as a signal. This is represented by the icon as <<Signal>> which is written above the event/action.
- **Self-transitions :** A state can have a transition that returns to itself, as in the diagram. This is most useful when an effect is associated with the transition.



Example : State diagram for login part of an online banking system.

- Logging in consists of entering a valid social security number (SSN) and personal id number, then submitting the information for validation.
- Logging in can be factored into four non-overlapping states: *Getting SSN, Getting PIN, Validating, and Rejecting*. From each state comes a complete set of **transitions** that determine the subsequent state.

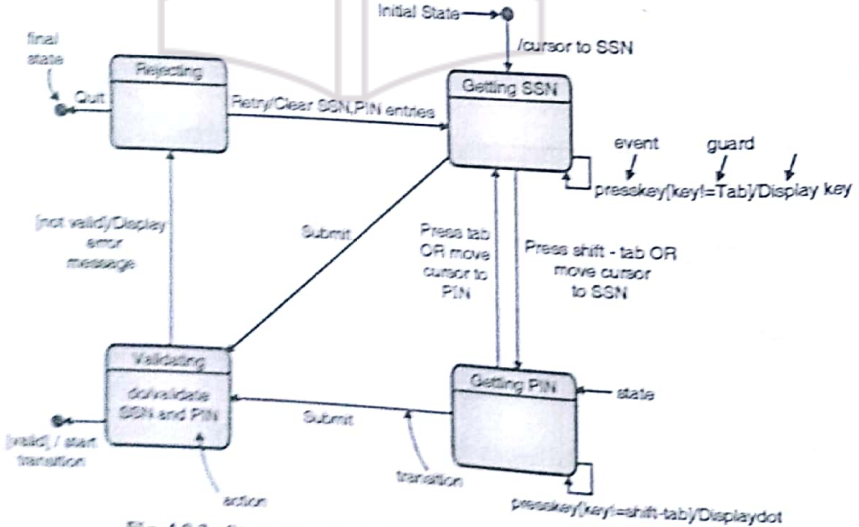


Fig. 4.8.3 : State diagram for login part of an online banking system

- **Actions inside the state:** Actions execute a function, assign a value to a data variable or initiate another transition. Specify these internal actions one per line. These internal actions (transitions) are processed without causing a state change.
- It takes the form: action-label / action-expression.
- The *action-label* may be any of the following:
  - **entry:** Executes the associated *action-expression* upon state entry.
  - eg. **entry / count:= 0; sum:= 0**
  - **exit:** Executes the associated *action-expression* upon state exit.
  - eg. **exit / ring bell.**
  - **do:** Executes the associated *action-expression* upon state entry and continues until state exit (or action completion).
  - eg. **do / display flashing light.**
  - **include**: The *action-expression* must name a finite automaton. The named automaton is a placeholder for a nested state diagram.
  - eg: **include / Order Processing**
  - *action-expression* - description of a computation.

Example : A state diagram for a temperature controller.

A temperature controller has been interface with a manufacturing unit in a factory which maintains the moderate temperature of 50%. If the temperature goes beyond or below the moderate temperature, the unit activities are cooling or heating units respectively and indicate through different signals.

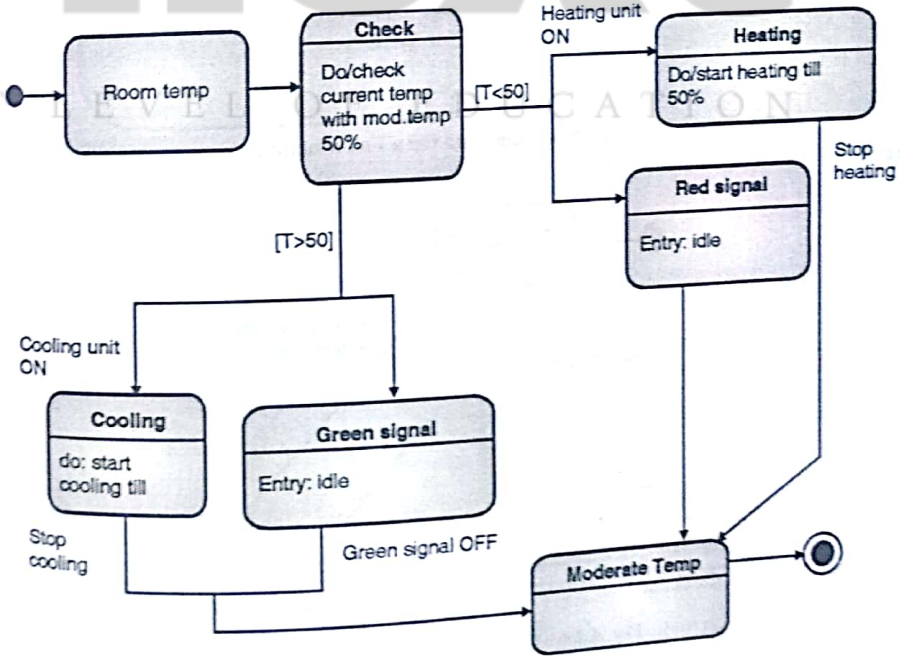


Fig. 4.8.4 : A state diagram for a temperature controller

Example : State diagram for a Fax Machine.

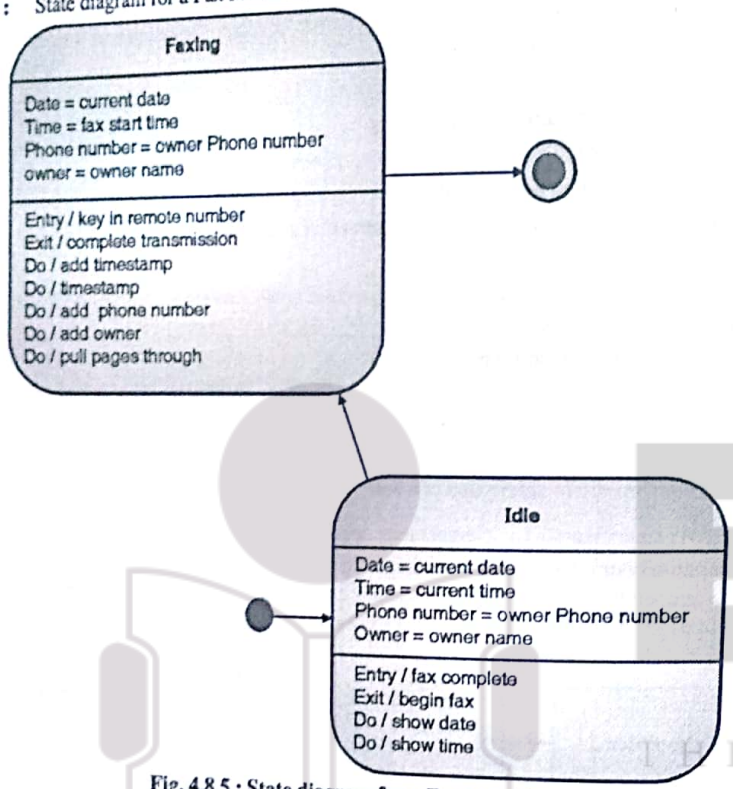


Fig. 4.8.5 : State diagram for a Fax Machine

Example : State diagram for ATM.

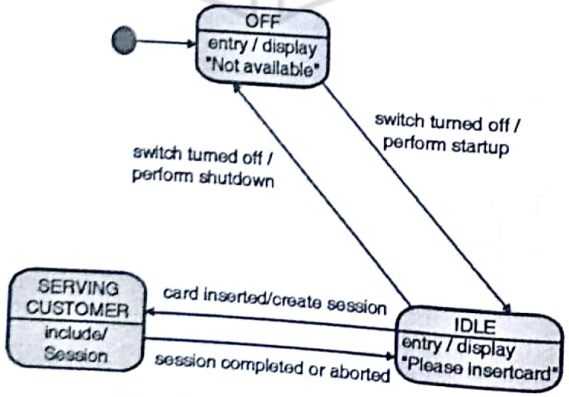


Fig. 4.8.6 : State diagram for ATM

Example : State diagram for Digital Watch.

A simple digital watch has a display and two buttons to set it, the A button and B button. The watch has two modes of operations display time and set time. In the display time mode, hours and minutes are displayed, separated by flashing colon. The set time mode has two sub modes, set hours and set minutes. The A button is used to select modes. Each time it is pressed the mode advances in sequence display, set hours, set minutes, display etc. Within the sub modes, the B button is used to advance the hour or minutes once, each time it is pressed. Button must be released before they can generate another event. Prepare a state diagram for digital watch.

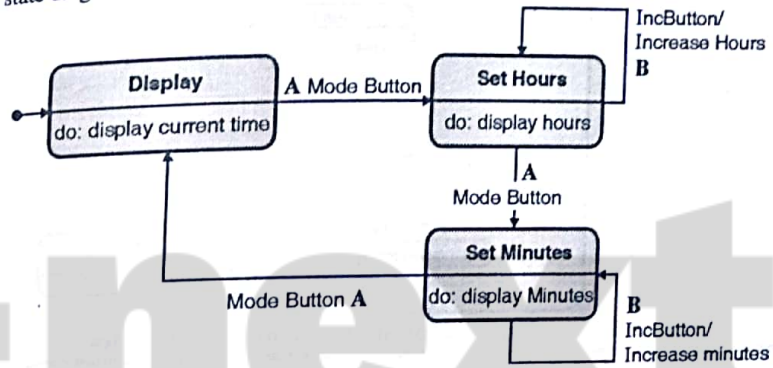


Fig. 4.8.7 : State diagram for Digital Watch

- **Choice Pseudo-State:** A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The Fig. 4.8.8 shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.

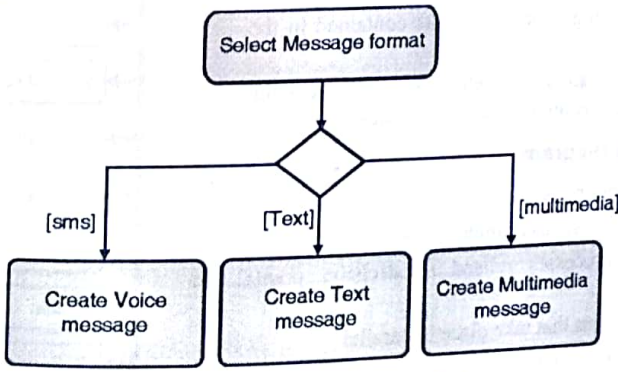


Fig. 4.8.8 : Choice pseudo-State

- **Nested States :** Nested states represents substates. The substate inherits the transitions of its superstate. Even though it affects the main state, a sub state is not shown as a part of the main state. Hence, it is depicted as contained within the main state flow.

Example : State diagram of phone line showing nested states in the main state *'Active'*.

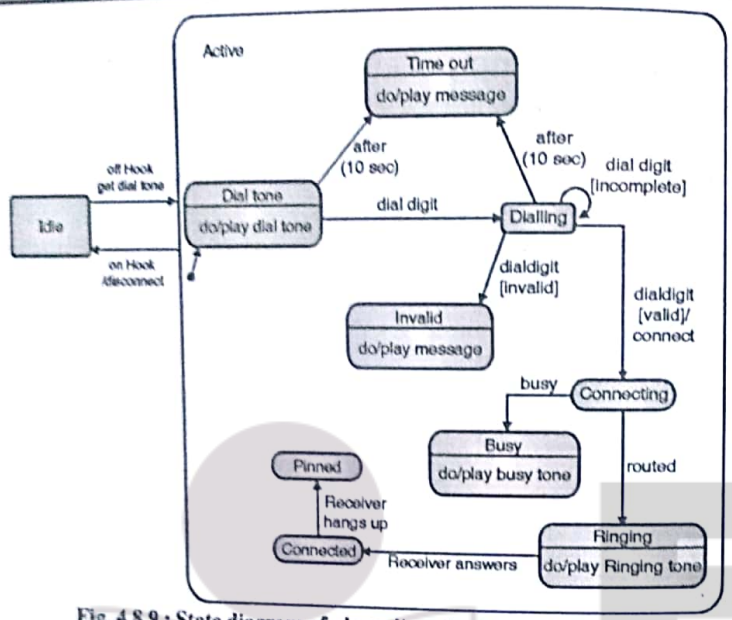


Fig. 4.8.9 : State diagram of phone line showing nested states

Syllabus Topic : Activity Diagram

4.9 Activity Diagram

- Activity diagram displays the sequence of activities.
- Activity diagrams show the workflow from a start point to the finish point detailing the number of decision paths that exist in the progression of events contained in the activity.
- They also describe situations, where parallel processing may occur in the execution of some activities.

Use of Activity diagram

- Describes Business rules
- Describes Complex series of multiple use cases
- Describes the processes related to decision points and alternate flows
- Describes operations that take place in parallel
- Describes software flows and logic control structures

Designing an Activity diagram

1. Initial node

The filled in circle is the starting point of the diagram. This indicates the beginning of the sequence of activities. An initial node makes it easier to read the diagram.

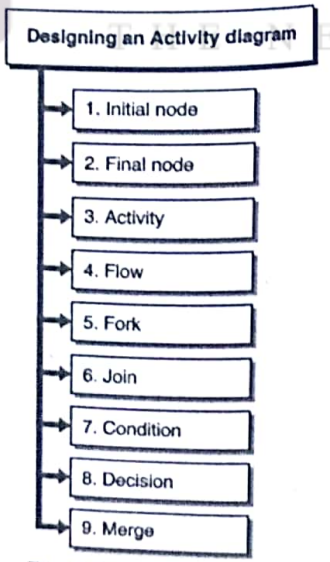


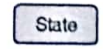
Fig. C4.7 : Designing an Activity diagram

2. Final node

The filled circle with a border is the ending point. An activity diagram can have zero or more activity final nodes. The final state ends the sequence of activities.

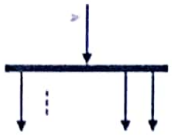
3. Activity

The rounded rectangles represent activities that occur. An activity may be physical, such as *Inspect Forms*, or electronic, such as *Display Student Screen*. It describes the action state of the system.



4. Flow

The solid arrows in the diagram represent the flow of activity. The outgoing arrow attached to an activity symbol indicates the transition i.e. triggered by the completion of the activity.

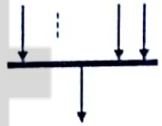


5. Fork

A thick black bar with one flow entering into it and several leaving it. This denotes the *beginning of parallel activity*. Forks have *One Entry Transition*.

6. Join

A thick black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue. This denotes the *end of parallel activity*. Joins have *One Exit Transition*. Is exactly opposite to 'Fork'.

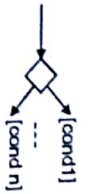


7. Condition

Text such as *[Incorrectpassword]* on a flow, defining a guard which must evaluate to true in order to traverse the node.

8. Decision

A *diamond with one flow entering and several leaving*. The flows leaving include conditions although some modelers will not indicate the conditions if it is obvious.



9 Merge

A *diamond with several flows entering and one leaving*. The implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow. Is exactly opposite to 'Decision'.

Example : Showing Decision, Condition and Merge.

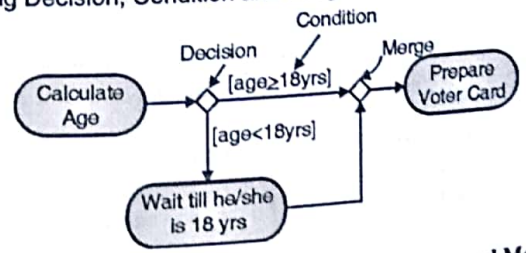


Fig. 4.9.1 : Example showing Decision, Condition and Merge

Partition (swimlanes) : Objects are generally organized into partitions, also called swimlanes, indicating who/what is performing the activities.

Example : Activity diagram for Aerobics Training Enrolment.

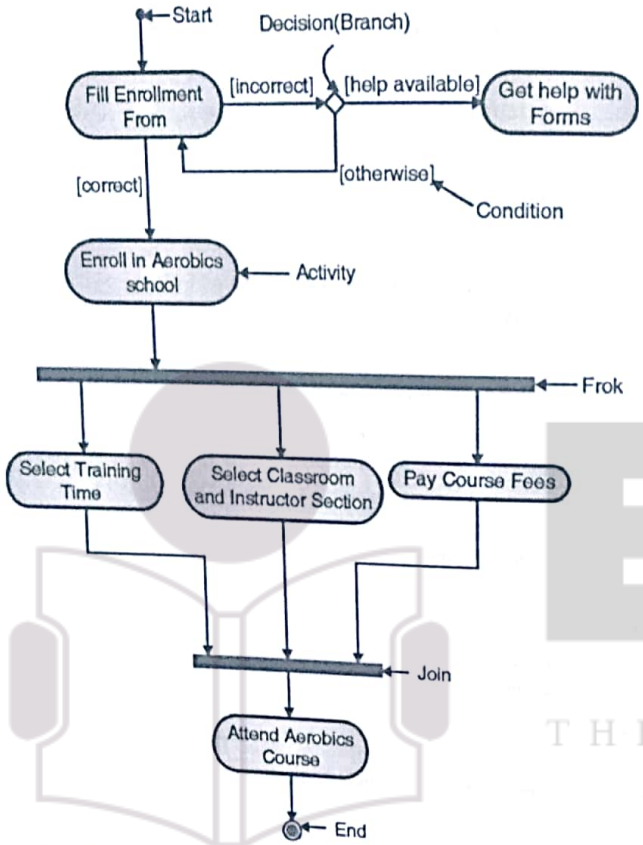


Fig. 4.9.2 : Activity diagram of Aerobics Training Enrolment

- After the 'Enrol in Aerobics School' activity completes, three activities occur: select training time, select classroom and instructor section, pay course fees. The **fork** (complex transition) indicates that these three activities can occur in any order.
- Then the **join** bar indicates that the 'Attend Aerobics Course' activity does not begin until after all the three activities (select training time, select classroom and instructor section, pay course fees) are complete.
- The two outgoing transitions of the 'Fill Enrolment Form' activity reflect a **decision** about whether the form is filled [correct]. The [incorrect] transition leads to a second decision, represented by the decision diamond, about whether to take help or fill the form again.

Example : Activity diagram for Login.

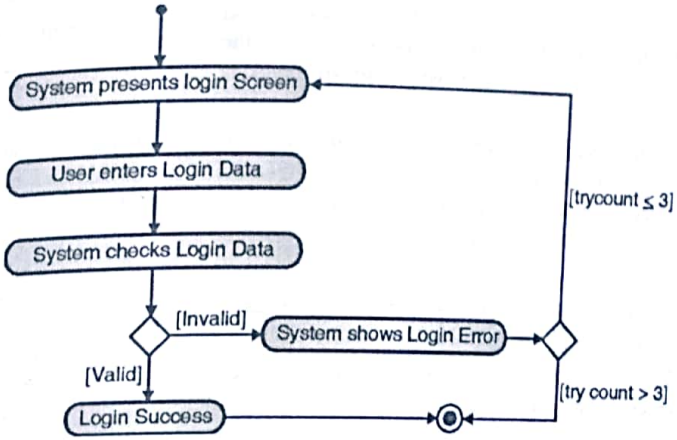


Fig. 4.9.3 : Activity diagram for Login

Example : Activity diagram for Customer process in Online buying of Toys.

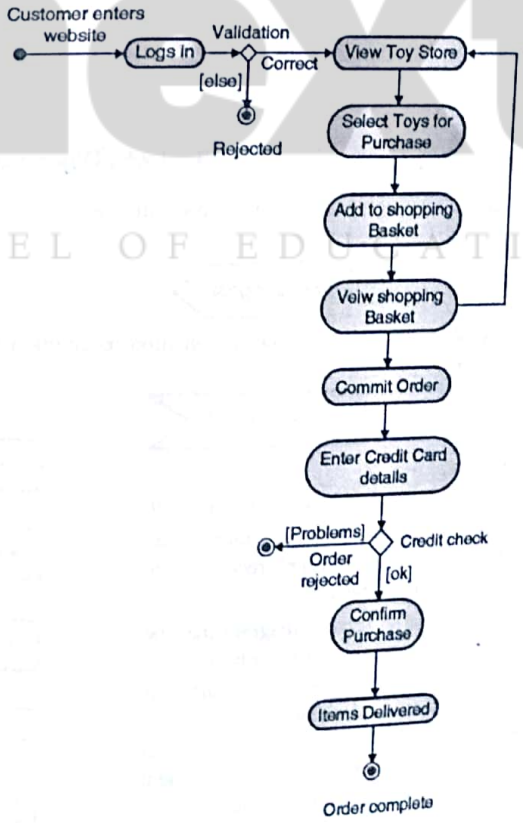
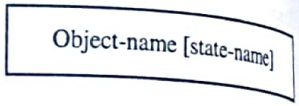


Fig. 4.9.4 : Activity diagram for Customer process in Online buying of Toys

- **Object State :** An object (indicated with a simple rectangle) in activity diagram represents instances of a specified object in a specified action state. If you specify the object name, the UML suite automatically encloses it within square brackets represented as;



- **Object flow :** The dashed arrows are used to represent the flow of objects. This shows how object states are used by action states. If an *action state generates an object* and it is used by the subsequent action state, then you can omit the transition (solid arrow) between the two states. You can use only dashed arrows. And these arrows are not labeled.

Example : Objects in states and object flows

In this example, completing the '*take order*' activity creates an '*Order*' object in the *[taken]* state.

The '*fill order*' activity takes an '*Order*' in the *[taken]* state and creates an '*Order*' in the *[filled]* state.

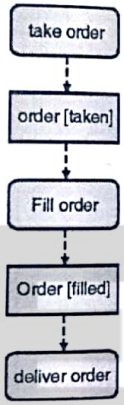
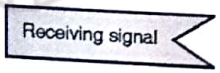


Fig. 4.9.5 : Object states and object flows

- **Signal sending :** A signal sending symbol shows a transition sending a signal. Do not label such signal sending transitions.

- **Signal receipt :** A signal receipt symbol shows a transition receiving a signal. Do not label such signal receiving transitions.



Example : Signal sending and receipt

In this example, the transition from '*take order*' to '*fill order*' sends an '*orderTaken*' signal to the '*Order*' object. The transition from '*fill order*' to '*deliverOrder*' receives an '*orderFilled*' signal from the '*Order*' object.

- **Swimlanes :** The contents of an activity diagram may be organized into partitions using solid vertical (or horizontal) lines also called as swimlanes. Each swimlane represents an organizational unit of some kind. A swimlane is a way to group activities performed by the same actor. You label each swimlane with the name of the class or department responsible for the activity states in the swimlane.

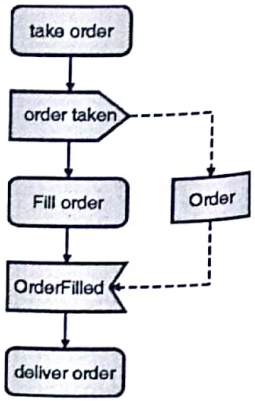


Fig. 4.9.6 : Signal sending and receipt

Example : Showing Swimlanes

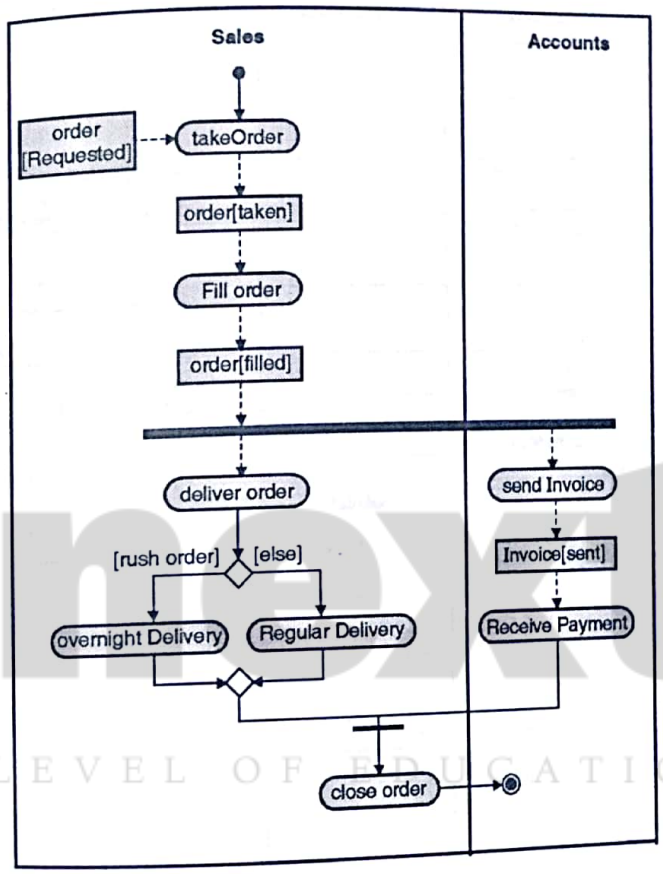


Fig. 4.9.7 : Showing Swimlanes

Example : Activity diagram to withdraw money from a bank account through ATM.

- The three involved classes of the activity diagram are *Customer, ATM,* and *Bank.*

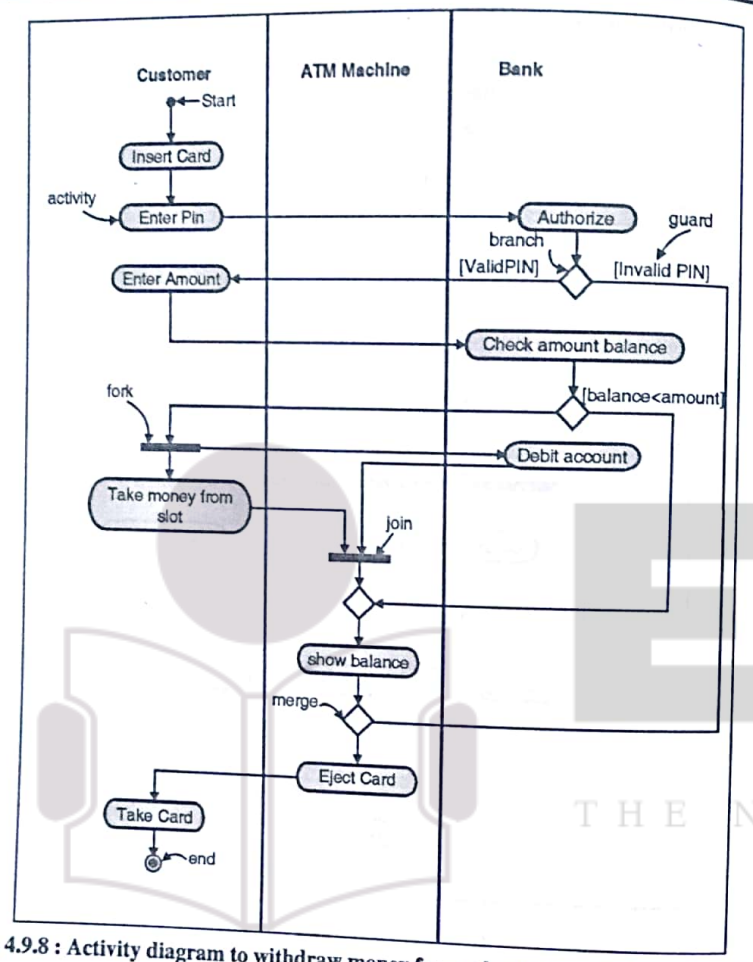


Fig. 4.9.8 : Activity diagram to withdraw money from a bank account through ATM.

4.10 Implementation Diagram

- Architecture model includes implementation diagram which shows the implementation phase of system development and its architecture and architecture. It describes the physical and logical structure.
- Physical architecture deals with
  - Detail description of the system and decompositions in terms of hardware and software.
  - Physical location of the classes and objects in which processes, programs and computers.
  - Dependency between different code files and connection hardware device.
- Logical architecture deals with
  - Functionality of the system
  - Functionality of the system deliver

  - Relationship among classes
  - Class and object collaboration to deliver the functionality.
- In short, Implementation deals with not only the code of the program but also with its implementation structure.

Examples of Implementation diagram :

1. *Component diagram* 2. *Deployment diagram*

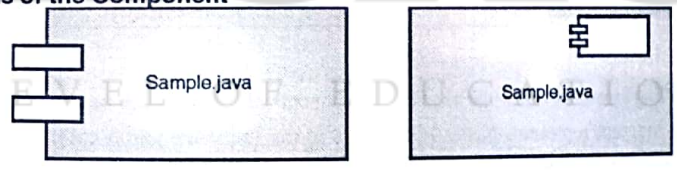
Syllabus Topic : Component Diagram

4.10.1 Component Diagram

Component

- A component is a distributable unit of software.
- A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
- A component is a subtype of Class which provides attributes and operations and participates in Associations and Generalizations.
- A Component may form the abstraction for a set of realizing Classifiers that realize its behaviour.
- In addition, because a Class itself is a subtype of an Encapsulated Classifier, a Component may optionally have an internal structure and own a set of Ports that formalize its interaction points.
- Components are concerned with the physical aspect of the system. Physical aspects are the elements like executable, libraries, files, and documents etc. which reside in the node.

Notations of the Component



Notation 1 Notation 2

Notation 1 is somewhat older notation used by UML.

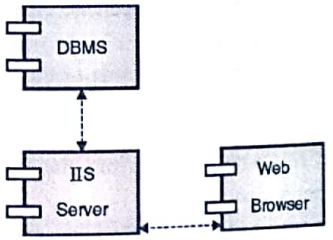


Fig. 4.10.1 : Component diagram of a 3-tier architecture of an application

- Every component has different name and properties of each component is different from each other.

- We use simple string to assign the name to a Component diagram. Sometimes the pathnames or the property parameters are also added to component name and in that case, the extendable name of component is used.

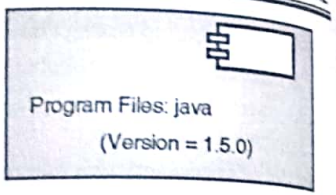


Fig. 4.10.2 : Extendable Notation

Similarities between Class and Component

- There are lot of similarities between a class and a component such as:
- Both have a distinct name
- Both realize a set of interfaces
- Both participate in dependency, generalisation and association relationship (remember class diagram)
- Both may have instances
- Both may be nested
- Both may participate in interaction.

Differences between Class and Component

Components	Classes
Components represent physical aspects related with world of bits	Classes represent logical abstraction
Physical packaging of logical components is at different level of abstraction.	Classes may not
Components only have operations that are reachable only through their interfaces.	Classes may have attributes and operations directly.

Component Diagram

- Component diagram shows how the physical components of a system are organised.
- Component diagram represents pieces of software in the implementation environment. Where the class and packages diagrams models the implementation view.
- According to the definition of the component, component is related with the physical aspects. Component diagrams are used to visualise the organisation and relationships among components in the system. These diagrams are also used to make executable system.
- Component diagram is somewhat different from the other diagrams which are used in UML. If we consider a simple class diagram, class contains the name of the class, attributes and the operation (function) respective to that class. In general class provide overall functionality. Component diagram never provide the functionality. But the component are those functionality description make by the component diagram. This is the futuristic importance of the component diagram.
- Here the components are files, libraries, packages etc.
- Component diagram can also be described as a static implementation of view of a system. Static implementation represents the organisation of the component at a particular movement.

Use of Component Diagram

- Visualize the components of a system.
- Describe the organization and relationships of the components.
- Construct executables by using forward and reverse engineering.

Designing Component Diagram

- Component diagrams are used to describe the physical artifacts of a system. We know that this artifact includes files, executables, libraries etc.
- So the purpose of this diagram is different, Component diagrams are used during the implementation phase of an application. But it is prepared well in advance to visualize the implementation details.
- Initially the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation.
- This diagram is very important since without it the application cannot be implemented efficiently.
- A well prepared component diagram is also important for other aspects like application performance, maintenance etc.
- So before drawing a component diagram the following artifacts are to be identified clearly:
  1. What kinds of Files are used in the related system.
  2. Which Libraries and other artifacts relevant to the application.
  3. Relationships among the artifacts (files and libraries).
- Rules to draw the component diagram:
  1. Use a meaningful name to identify the component for which the diagram is to be drawn.
  2. Prepare a mental layout before producing using tools.
  3. Use notes for clarifying important points.
- Component diagram with following object oriented modeling language is used for management of these files which represent the work product components.

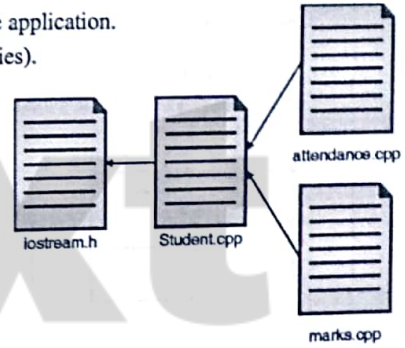


Fig. 4.10.3 : Modeling Source Code

Example : An employee of a company stays in a flat or bungalow. It may be owned by the company, employee or may be rented by either the company or the employee. The owner who lends houses to company may themselves be company employee. The Tenant in a flat has to pay additional water and electricity charges.

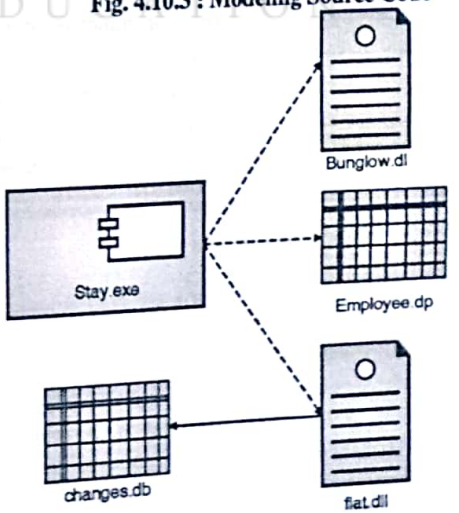
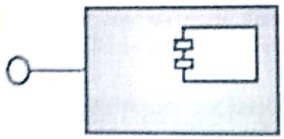
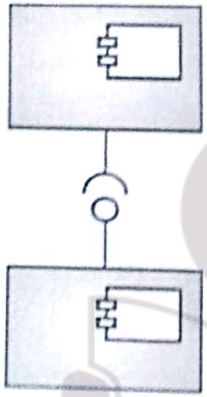


Fig. 4.10.4 : Modeling Physical Databases

Other Notations used in Component Diagram

Notation	Use
	Components implement interfaces
	Assembly Connector

Advantages of Component Diagram

- Models the real software in the implementation environment.
- They make known software configuration issue through the dependency relationships.
- It provides the accurate picture of existing system prior to making changes or development.
- They can reveal bottlenecks in an implementation without forcing us to read all of the code.

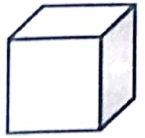
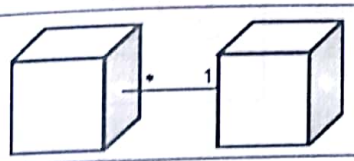
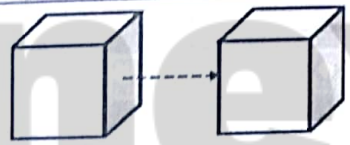
Syllabus Topic : Deployment Diagram

4.10.2 Deployment Diagram

- Deployment diagram show the configuration of run-time processing elements and the software components, processes and objects that live in them.
- Deployment diagram models the hardware of the implementation environment.
- A deployment diagram is a graph of nodes connected by communication association. In general the deployment diagram describes the run time architecture of processors, devices and the software components. Each of these node in the deployment diagram typically represents one of the hardware part like disk drive, client machine, server machine, processor, networking structure etc.
- Here, node means the components or logical elements which are used at the run time. Basically that are class, objects or collaborations which are executes at run time.
- That node connects for each other for communication purpose.

- Connection between nodes shows the system interaction path.
- Collaboration diagram models the topology of the hardware. It also models the embedded system. It put client-server model and distributed application model hardware.

Table 4.10.1 : Notations used in Deployment Diagram

Sr. No.	Notation	Use
1.		Node: A node is drawn as a three dimensional cube with names inside it.
2.		Association between two nodes: Communication association connects to each other.
3.		Dependency

Advantages of Deployment Diagram :

1. Models the hardware platform for the system.
2. Identifies hardware capabilities that affect performance planning and software configuration.

Example : Database server (any kind of) can connect with client machine through Internet

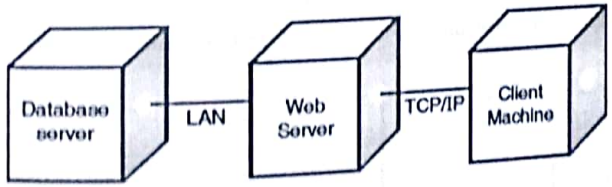


Fig. 4.10.5

Example : Online Shopping for checking the products

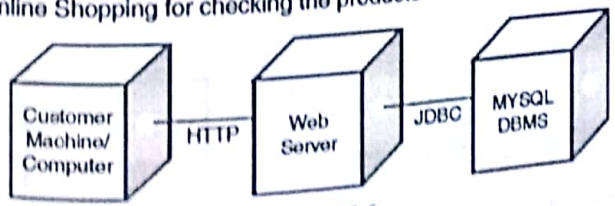


Fig. 4.10.6

1. We can combine the component diagram with deployment diagram as follows.

Example

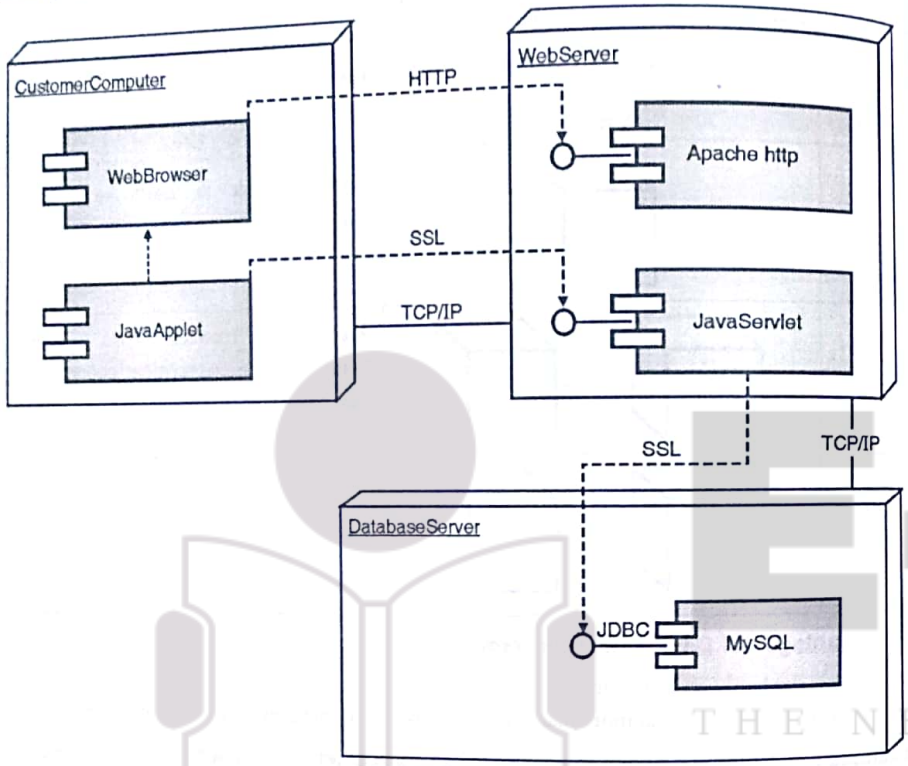


Fig. 4.10.7

Example : A simple deployment diagram of 3-tier architecture of an application.

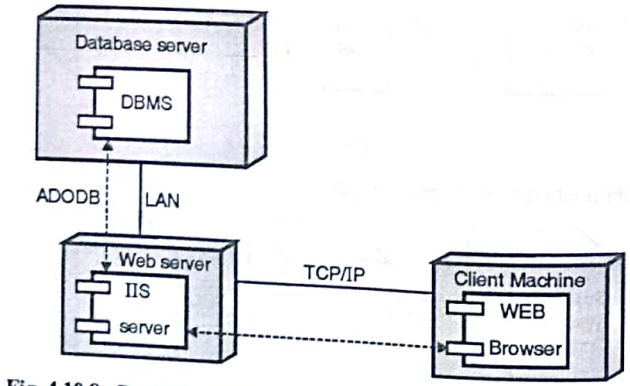


Fig. 4.10.8 : Deployment diagram encompassing component diagram

Review Questions

Q. 1 "UML is mainly used for software system". State 'True' or 'False' with explanation

Q. 2 What is classification theory?

Q. 3 List and explain the stereotypes used for modeling and interactions among objects.

Q. 4 Prepare use case diagram for bus reservation system.

Q. 5 Draw a class diagram for "mobile company". They have different distributors at different areas. Different facilities are provided such as incoming calls free, sending E-mails, songs can be seen etc. (Make necessary assumptions).

Q. 6 An employee of a company stays in a flat or bungalow. It may be owned by the company, employee or may be rented by either the company or the employee. The owner who let houses to company may themselves be company employee. The Tenant in a flat has to pay additional water and electricity charges. Draw component and deployment diagram for the system.

Q. 7 Draw a collaboration diagram for Railway reservation system. The passenger is required to fill a reservation from giving details of his journey. The counter clerk ensures whether the place is available and prepares a booking statement.

❑❑❑

UNIT II

CHAPTER 5

System Design

Syllabus

System/Software Design, Architectural Design, Low-Level Design Coupling and Cohesion, Functional-Oriented Versus The Object-Oriented Approach, Design Specifications, Verification for Design, Monitoring and Control for Design.

5.1 Introduction

This chapter focuses on the design phase of SDLC. After requirement gathering and analysis phase, the next step is to *design the analysis model*.

System

System is an interrelated set of components, with identifiable boundaries working together for achieving some goal.

A system has nine characteristics

1. **Components:** These are the indecomposable or irreducible parts that make up a system which are also called as subsystems.
2. **Inter-related Components:** It is about the dependency between the subsystems i.e. dependency of one subsystem on one or more subsystems.
3. **A boundary:** It is the line that marks the inside and outside of a system and that sets off the system from its environment.
4. **A purpose:** It is the overall goal or function of a system.
5. **An environment:** It encompasses all the external entities of a system that interact with the system.
6. **Interfaces:** Interaction between the system and its environment or interaction between various subsystems.
7. **Input :** Whatever a system takes from its environment in order to fulfil its purpose.
8. **Output:** Whatever a system returns to its environment in order to fulfil its purpose.
9. **Constraints:** The limit to which a system can accomplish or the restrictions on developing the system.

Syllabus Topic : System/Software Design

5.2 System Design

Systems analysis : Process of studying an existing system to determine how it works and how it meets user needs.

Systems design : Process of designing a plan for an improved system based upon the results of the existing system analysis.

Need of System Design

- It is an Effective method to solve problems of existing system as it designs the new plan based upon the study of existing system analysis.
- It differentiates logical view and physical view of the software to be built up.
- Designing also partitions the requirements such that it becomes easy to build up a model and develop its documentation.
- It keeps track of all interfaces and designs them properly.
- Whenever possible, graphics are used to design the analysis model.

Main Objectives of Design

- Practicality – Efficiency – Cost – Flexibility – Security

Analysis & Design Model

- Analysis modelling uses diagrammatic form and text to describe requirements of :
  - Data o Functions o Behaviour of o The software to be built
- It should have the following features :
  - Easy to understand. o Straightforward to review. o Correctness.
  - Completeness. o Consistency.

Who does it ?

- Software engineers or – System analysts or – Project manager or – Modeller

Importance

- Analysis model is multidimensional.
- Design phase completely depends on analysis model.
- If some deficiency remains in the analysis models then errors will be found in product to be built.

Ensures

- It must be review for correctness, completeness and consistency.
- It should ensure that it accomplishes all needs of stakeholders.
- It establishes a foundation from which design can be conducted.

5.2.1 Thumb Rule

Arlow and Neustadt produce rules of thumbs for the analysis model. These rules are given below:

1. Model should focus on requirement visible in business domain.
2. Each element of the analysis model should add to all requirements and provides insight into information domain function and behaviour of the system.
3. Some functional and non functional models are required to design the software.
4. Minimize coupling throughout the system. It is important to represent relationship between classes and functions. That means functions are interconnected to each other very tightly then efforts are put to reduce this interconnectedness.
5. It gives assurance that analysis model provides value to all stack holders.
6. It keeps model as simple as it can be.

5.2.2 Design Modeling Principles

Design model provides a variety of different views of the system just like architecture's plan for a house. Different methods like data driven, pattern driven or object oriented methods are used for constructing the design model. And all these methods use a set of design principles for designing a model.

1. **Design must be traceable to the analysis model** Analysis model represents the information, functions and behaviour of the system. Design model translates all these things into architecture: a set of sub systems that implement major functions and a set of component level designs that are the realization of analysis classes. This implies that design model must be traceable to the analysis model.
2. **Always consider architecture of the system to be built** Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, program control flow and behaviour, the manner in which testing is conducted, maintainability of the resultant system and much more.
3. **Focus on the design of data :** Data design encompasses the manner in which the data objects are realized within the design. It helps to simplify the program flow, makes the design and implementation of software components easier and makes overall processing more efficient.
4. **Interfaces (both user and internal) must be designed :** The data flow between the components decides the processing efficiency, error flow and design simplicity. A well designed interface makes integration easier and tester can validate the component functions more easily.
5. **User interface should consider the user first :** The user interface is the main thing of any software. No matter how good its internal functions are or how well designed its architecture is but if the user interface is poor and end users don't feel ease to handle the software then it leads to the opinion that the software is 'bad'.
6. **Component level design should exhibit functional independence :** It means that the functions delivered by a component should be cohesive i.e. it should focus on one and only one function or sub function.
7. **Components should be loosely coupled :** Coupling (Union or Combination) of different components into one is done in many ways – via a component interface, by messaging or through global data. As the level of coupling increases, error propagation also increases and overall maintainability of the software decreases. Thus, component coupling should be kept as low as possible.
8. **Design representations should be easily understood :** The design model helps engineers to generate code, testers to test software, and maintain the software easily in the future. If the design is difficult to understand, it will not serve as an effective communication medium.
9. **The design model should be developed iteratively :** Designs are done in iterations to make designing as simple as possible.

Syllabus Topic : Architectural Design

5.3 Architectural Design (High-level Design)

Architecture design of a system or a software constitutes of a complete framework that includes the structure – its components and how they are integrated with each other to form a complete system.

Need of Architectural design

- It evaluates all top-level designs.
- Highlights early design decisions by specifying the functional and performance behaviour of the system that leads to ultimate success of the system
- It constitutes of a relatively small and easily understandable model of how the system is structured and how its components are integrated to work together. This enables communication between all stakeholders of the system

Architectural Design Representations

1. **Structural model :** represents the ordered collection of program components
2. **Dynamic model :** shows the behavioral aspect of the software and indicates how the system configuration changes with the change in the functions.
3. **Process model :** depicts the design of the business or technical process that needs to be implemented in the system

Functional model: Represents the functional hierarchy of a system

Framework model: shows increase in the level of abstraction.

Outputs of Architectural Design

1. **Reports :** audit report, progress report, and configuration status accounts report
2. **Plans :**
   i) Software verification and validation plan
   ii) Software configuration management plan
   iii) Software quality assurance plan
   iv) Software project management plan.

Architectural Styles

1. Data centered architecture

This type of architecture has two distinct components - a central data structure known as data store (central repository) and a collection of client software.

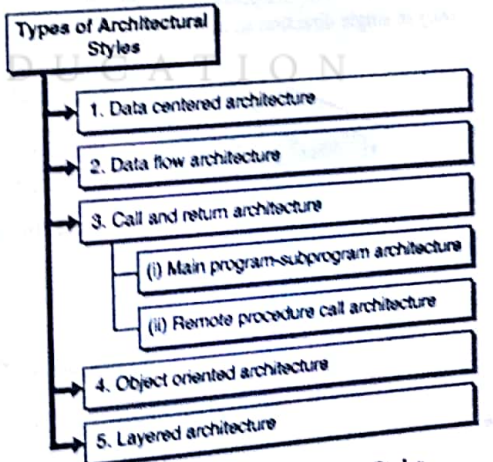


Fig. C5.1 : Types of Architectural Styles

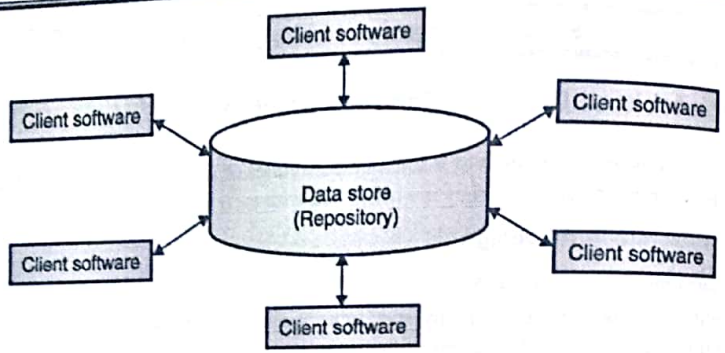


Fig. 5.3.1 : Data centred architecture

Advantages

- Client software operate independent of each other
- Central repository is independent of client software
- Scalable : new clients can be added easily
- Easily modifiable

2. Data flow architecture

This type of architecture is used for the systems that accept inputs and transform it into the desired outputs by applying a series of transformations. Each component called as filter transforms the data and sends it to other components (filters) for further processing using the connector known as pipe. Each filter works independent of each other and each pipe is a unidirectional channel that works only in single direction so as to pass the data from one filter to another filter.

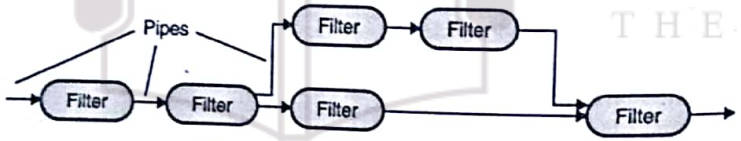


Fig. 5.3.2 : Data flow architecture

Sr. No.	Advantages	Drawbacks
1.	Supports reusability	Does not provide enough support for applications requiring user interaction.
2.	Maintainable and modifiable Supports concurrent execution	Difficult to synchronize two different streams

3. Call and return architecture

This type of architecture designs a program structure that can be easily modified. This architecture consists of following two sub styles.

(i) Main program-subprogram architecture:

The main function is decomposed to invoke number of program components which in turn invoke other components.

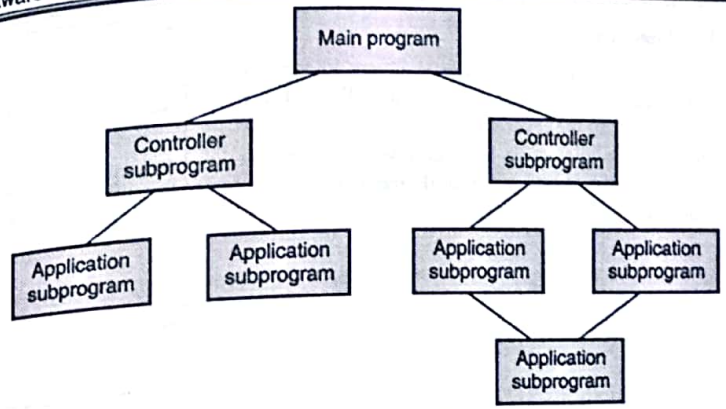


Fig. 5.3.3 : Main program-subprogram architecture

(ii) Remote procedure call architecture:

Components of the main program or subprogram architecture are distributed over a network across multiple computers.

4. Object oriented architecture

This type of architecture designs the components known as **objects** of the system such that it encapsulates data and operations (used to manipulate the data) of the object. In this style, the objects interact with each other through methods known as **connectors.**

Advantages

- Maintains the integrity of the system
- Allows to decompose the problem into a collection of independent objects
- Encapsulation allows to hide the implementation details of one object from other objects. Therefore, every object can be changed without effecting other objects.

Object Oriented architecture constitutes of 4 types of modelling elements : 1

- Scenario based elements.
- Flow oriented elements.
- Class based elements.
- Behavioural elements.

The specification of each element may differ from project to project.

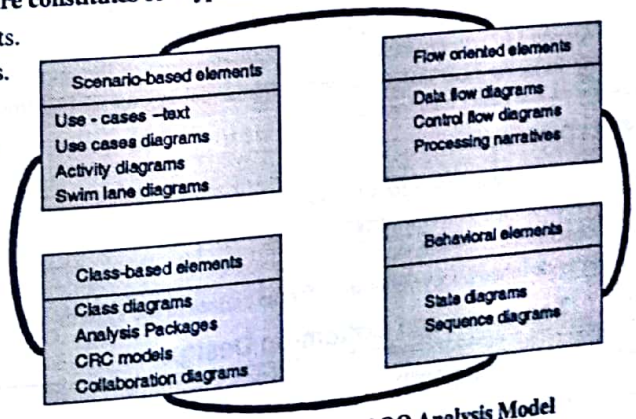


Fig 5.3.4: Different types of OO Analysis Model

5. Layered architecture

This type of architecture consists of several layers known as components arranged in a hierarchical manner. Each layer provides a set of services to the layer above it and acts as a client to the layer below it. The interaction between layers is provided through protocols known as connectors that define a set of rules to be followed during interaction.

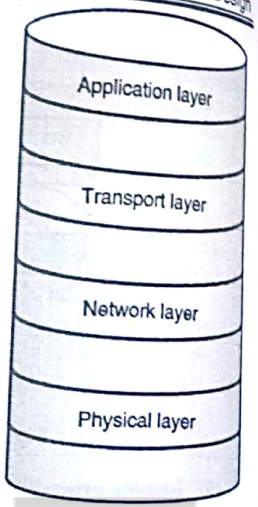


Fig. 5.3.5: Example of Layered architecture – OSI layered architecture

Syllabus Topic : Low-Level Design

5.4 Low-Level Design

- High-level design (HLD) identifies the system's general environment (hardware, operating system, network, and so on) and architecture (client/server, and service-oriented). It works on modules such as reporting modules, databases, and top-level classes.
- Low-level design (LLD) provides extra details that are necessary before developers can start writing code. It tells how the parts of the system will work together and refines the definitions internal and external interfaces.
- High-level designs focus on *what*. Low-level designs focus on *how*.
- HLD designs the overall architecture of the entire system from main module to all sub module. LLD defines Internal logic of corresponding sub modules.

A good LLD document helps in easy development of the application. The code can be developed directly from the LLD document with minimal debugging and testing. Other advantages of LLD are lower cost and easier maintenance.

LLD document

- Provides the interface for all classes
- Describes the data structures
- Describes the algorithms
- Provide inheritance relationships for the classes in the system

5.5 Top-down and Bottom-up Design

Top-Down Approach

- It is also known as step-wise design and is about breaking down of a system to get details of its compositional sub-systems.

- In a top-down approach, the overview of the system is first formulated which specifies but does not give any detailing of first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements.
- A top-down model is usually specified using the 'black boxes' which make it easier to manipulate.
- Top-down approach focuses on planning and a complete understanding of the system. It strictly tells that no coding can begin until a sufficient level of detail has been reached in the system design of at least some part of the system.

Advantages of Top-Down Approach

- Separating the low level work from the higher level abstractions leads to a modular design where development of sub systems become self contained i.e. independent of each other.
- Illustrates the integration of low level modules.
- Reduces errors because each module has to be processed separately, so programmers get lot of time for processing.
- It is less time consuming because each programmer is only involved in one part of the big project.
- Each programmer has to apply his knowledge and experience to only his module and so the project becomes more optimized.
- Easy to maintain because if an error occurs in the output, it is easy to identify the source of the error i.e. it is easy to find that which module of the entire program has caused that error.

Bottom-Up Approach

- It is about bringing together of sub systems to give rise to a better system, thus making the original systems as the sub-systems of the emergent system.
- In a bottom-up approach, individual base elements of the system are first specified in great detail. These elements are then integrated together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed.
- Bottom-up emphasizes coding and early testing, which can begin as soon as the first module has been specified. This approach, however, runs the risk that modules may be coded without having a clear idea of how they link to other parts of the system.

Syllabus Topic : Functional-Oriented Versus The Object Oriented-Design Approach

5.6 Function Oriented vs. Object Oriented Design

- Jim Rumbaugh and his Co-workers found out the method of the Object Modeling Technique (OMT) which describes the analysis, design and implementation by using OMT.
- Rumbaugh method further became very popular among various authors because of its user friendly nature and simplicity ways.
- Rumbaugh uses familiar symbols and techniques in his analysis and design.

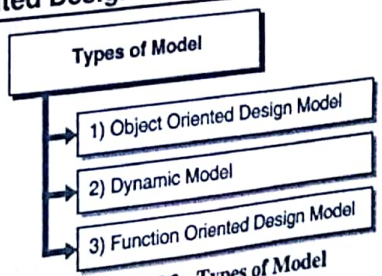


Fig. C5.2 : Types of Model

- Rumbaugh OMT provided three set of concepts which provide three different views of the system. The three models are as define below;

1) Object Oriented Design Model

Object Model describes the static structure of the objects in the system - identification of attributes, operations and their relationship.

The main concepts are;

- Class
- Attributes
- Operation
- Inheritance
- Association (i.e. relationship)
- Aggregation

- The Rumbaugh object model is very much like an entity relationship diagrams (ERD) except that there are no behaviors in the diagram and class hierarchies.
- Following are the notations used by the Rumbaugh method in his object model;

Table 5.6.1 : Notation used by Rumbaugh in his Object model

Sr. No.	Shape Used	Purpose
1	Class / Attributes / Operation	Class Representation
2.		One-to-one association
3		One-to-many association
4		Specialisation

Example

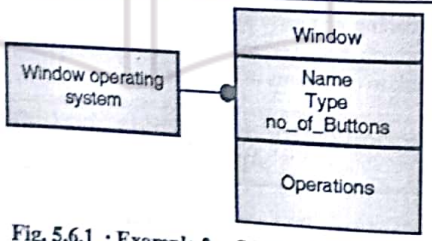


Fig. 5.6.1 : Example for OMT object Model

- OOD designs technical solutions (User Interface, Data Schemas, Object Classes, etc) to the problems identified by analysis phase.
- When we are solving problems using OOD, the following tasks must be accomplished by the user.
  - Basic user requirements are known to customer as well as software engineer
  - Design the classes according to requirements –Attributes and Methods are defined.
  - Hierarchy of the classes are defined.
  - Relationships among classes are defined.
  - Behaviour of object is decided.

  - All above tasks are repeated till model is not complete.

2) Dynamic Model

- The dynamic model is a "state transition" diagram that shows how an entity changes from one state to another state.
- The main aspects are;
  - State
  - Sub / Super State
  - Event
  - Action
  - Activity
- This kind of diagram states network of activity.
- Following are the notations which are used by the Rumbaugh method in his dynamic model;

Table 5.6.2 : Notation used by Rumbaugh in his Dynamic model

Sr. No.	Shape Used	Purpose
1		Start
2.		States
3		Transition

Example

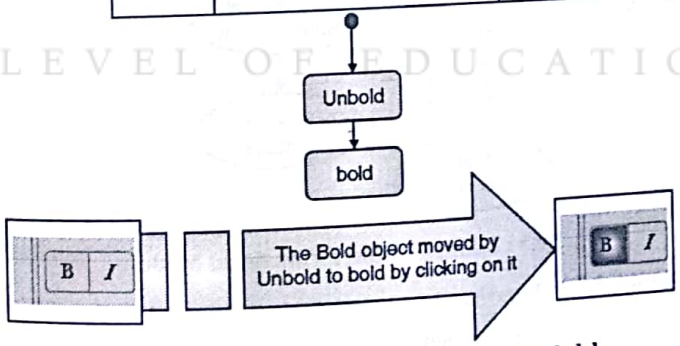


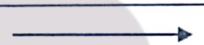
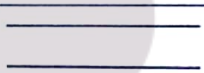
Fig. 5.6.2 : Example of OMT Dynamic Model

3) Function Oriented Design Model

- The functional model is the equivalent of the familiar Data Flow Diagrams (DFD) from a traditional systems analysis.
- This model describes the data value transformation within the system.
- A *function model* in systems engineering and software engineering is a structured representation of functions, behaviors, activities or processes within the modeled system.
- A function model, also called an activity model or process model, is a graphical representation of an enterprise's function within a defined scope.

- A well-known example of functional modeling language is data flow diagrams (DFD).
- The main concepts are :
  - Process
  - Data Store
  - Data Flow
  - Control Flow
  - Actor (Source / Sink)
- Following are the notations which are used by the Rumbaugh method in his functional model;

Table 5.6.3 : Notations used by Rumbaugh in his Function model

Sr. No.	Shape Used	Purpose
1.		Process
2.		Data Flow
3.		Data Store
4.		External Entity

Example

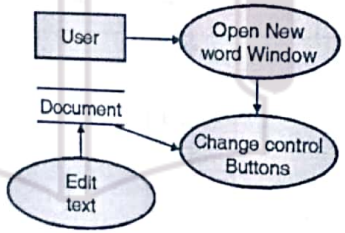


Fig. 5.6.3 : Example of OMT Functional Model

Table 5.6.4 : Function Oriented v/s Object Oriented Design

Sr. No.	Function Oriented Design	Object Oriented Design
1.	Focuses on functions which operate on data.	Focuses on objects which are conceptual entities having attributes and methods.
2.	It is a set of functions each of which performs a task.	It is a set of objects that can be modified and that can perform tasks.
3.	Relies on identifying functions which transform their inputs to create outputs.	Relies on encapsulation of objects and messaging between objects.

Sr. No.	Function Oriented Design	Object Oriented Design
4.	Divides a bigger problem set to small functional units and then organizes these functional units to design the solution.	Identifies the objects involved in the system and designs the solution based on their relationships and interactions.
5.	Used mainly for computation of sensitive applications.	Used mainly for evolving systems that mimic a business process.

Syllabus Topic : Design Specification

5.7 Design Specification

- Design specification enables to make planned decisions with broad consequences.
- Best design specification is implemented by user requirements as well as previous experiments.
- The design process starts from object oriented Analysis and ends with system design.
- Design specifications can be categorized into two broad categories .

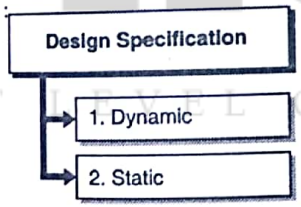


Fig. C5.3 : Design Specification

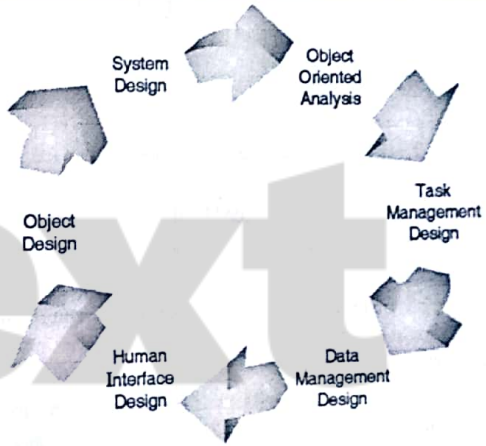


Fig. 5.7.1 : Process Flow of a Design

➜ **1. Dynamic**

- Data flow diagrams (DFDs)
- State transition diagrams
- Statecharts
- Structure diagrams

➜ **2. Static**

- Entity Relationship Diagrams(ERDs)
- Class diagrams
- Structure charts
- Object diagrams

- The design specification helps developers to take decision on how the problem will be solved.
- During this system design stage, it solves the problems which basically arises from the system analysis phase where developers think on :
  - Overall structure of the system
  - Subsystem of the system
  - Overall style of the system
  - Reuse of the system

5.8 Partitioning the Model

- The partitioning of the target system into subsystems, based on the combination of knowledge of the problem domain and the proposed architecture of the target system (solution domain).
- The breaking up of the system into the small pieces of the subsystems is the first step of the system design.
- Since the major piece which further implement is nothing but the sub system. That's nothing but the partitioning of the model.

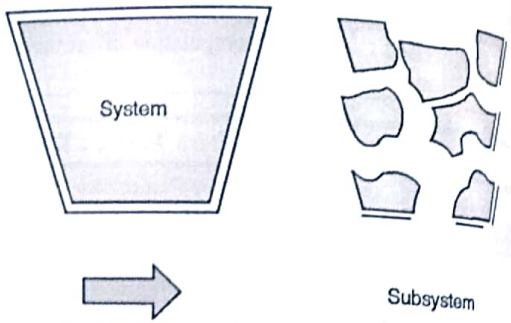


Fig. 5.8.1 : Partitioning the Analysis Model

- A subsystem is not an object nor a function but a group of classes, associations, operations, events and constraints that are interrelated and have a well defined and small interface with other subsystems.
- Whatever services provided by the subsystem according to that subsystems are identified.
- The subsystem should have a well-defined interface through which all communication with the rest of the system occurs.
- With the exception of a small number of "communication classes," the classes within a subsystem should collaborate only with other classes within the subsystem.
- The number of subsystems should be kept small. Subsystem may in turn be decomposed into smaller subsystem, of its own. The lowest level subsystems are called as modules.
- The decomposition of system into subsystems may be organized as a sequence of horizontal layers or vertical partitions.

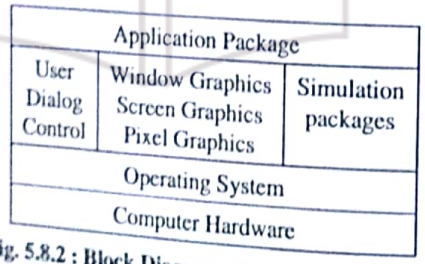


Fig. 5.8.2 : Block Diagram of typical application

- A subsystem can be partitioned internally to reduce complexity.

5.9 Abstraction and its Types

- Abstraction can be defined as an ability to look at something without being concerned with its internal details.
- This concept was introduced with structured programming where it is sufficient to know that a given procedure performs a specific task or not.

Abstraction Types

There are two types of Abstraction –

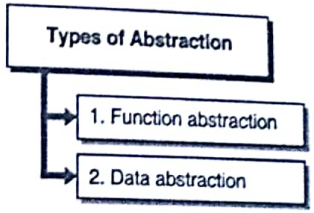


Fig. C5.4 : Types of Abstraction

1. Function abstraction

How is it done is not at all important as long as the procedure is reliable. This is known as ***Function abstraction***. For example, when you are using printf() function in C you are not concerned about how it is implemented but you know that it can be reliably used to output the data on the screen.

2. Data abstraction

It is used for data as functional abstraction for operations. With data abstraction, data structure and items can be used without having to be concerned about the extract details of implementation. For example, floating point numbers are abstracted in programming language.

- It means that one need not bother about the binary representation of floating point numbers gauges. It means that one need to know the details of binary multiplications in order to multiply two floating-point numbers.
- Similarly, a data type called fraction can be implemented using data abstraction.
- Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).
- We see that a tablet capsule is made up of two or more chemical materials. That chemical material is encapsulated in a cover. The same feature is adapted in OOD.
- Wrapping of data and functions into single unit (or class) is known as 'Encapsulation'. Data encapsulation is the most salient features of a class.
- The data is not accessible to the outside world. It is accessible by only those functions, which are wrapped in a class.

5.10 Modularity

- Modularity is about dividing the complex system into modules or components. The software is decomposed into separately named and addressable components.
- Modularity must be effective i.e. each module must focus on exclusively well-constrained aspects of the system i.e. it should be cohesive in its functions.
- Also, modules should be interconnected in a relatively simpler manner i.e. each module should show low coupling with other modules.
- A **module** is a system component that provides services to other components and is itself dependent on the services provided by other components. It is not normally considered as a separate system.

A system can be divided into different categories of modules or components :

Problem domain component

- The subsystem that is responsible for implementing customer requirements directly.
- Depending on the customer requirements, the design pattern knows the problem exactly.

Human interaction component

- This is a very important component of the OO design model since using this component, a model interacts with the Human.
- The subsystems that implement the user interface (this includes reusable GUI subsystems).
- For example Microsoft Excel has various cells in which we put entries and manipulate them. Each different entry is classified into different manner.

Table 5.10.1 : Example of Microsoft Excel Database

ID	Name	Salary
1	Nilesh	8000
2	Atul	5700
3	Sourabh	7000
4	Sunil	10000
		30700

- In Table 5.10.1 we make three columns ID, Name and Salary which are according to the class design. Here all of the three are attributes, Operations(Here we make sum of third column Salary) we use *sum* method for that.
- We also use alternative pattern in the OO Design.

Task management component

The subsystems that are responsible for controlling and coordinating concurrent tasks that may be packaged within a subsystem or among different subsystems;

Data management component

The subsystem that is responsible for the storage and retrieval of objects.

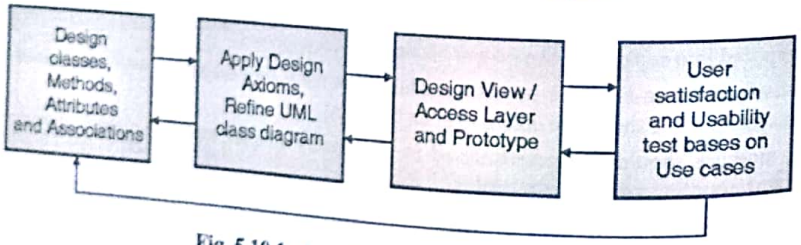


Fig. 5.10.1 : Looping within components

Advantages of Modularity

- By modularizing the system design, planning the development becomes more easy.
- Modules can then be developed in increments and delivered fast.
- Changes to modules can be easily accommodated as the changes in one module don't affect the other.

- Testing and debugging can be performed more effectively.
- Long-term maintenance can be performed by no serious side effects.

Syllabus Topic : Coupling and Cohesion

5.11 Coupling and Cohesion

Coupling

- Coupling within a software system is the degree to which each module depends on other modules. It is the degree of dependency among the components.

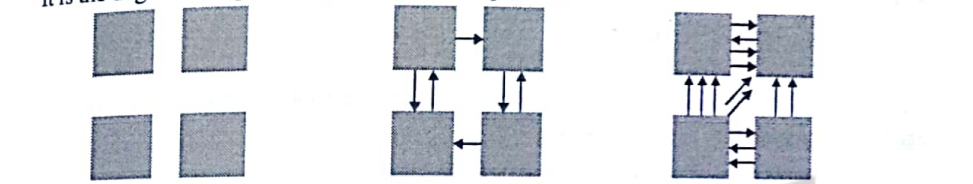


Fig. 5.11.1 : Various Types of Coupling

- Coupling is generally considered a bad thing where software components are dependent for some specific details of other components.
- In high coupling, the components are very closely and tightly tied to each other which make it difficult to modify the components of the system because modifying a component affects all other components to which the modifying component is connected.
- ***Therefore, Low-Coupling is good.***
- Object-oriented design has low coupling i.e. the objects (components) are independent of each other. Therefore, modifying one object doesn't have any effect on another object.

Example : Functions used in C++ & Java programming represent coupling i.e. dependency between software components (functions)

Cohesion

- Cohesion is the measure of internal interdependence with in a component i.e. the degree to which all elements of a component are directed towards a single task is called cohesion.
- ***Therefore, High-Cohesion is good.***
- Cohesion is considered as good thing because it is about creating a software component that combines related functionalities into a single unit. This is the core principle of OOD where it creates a component that encapsulates the objects of similar properties and functions that strive to perform same single task.

Example : Classes and objects in C++ & Java programming represent cohesion i.e. the degree to which the elements of a single component form a meaningful unit (object) and perform similar task.

- Cohesion represents how tightly the internal elements of a module are bound to one another - Coupling is degree of independence between different modules.
- High cohesion and low coupling is main criteria for good s/w design.

5.12 Structure Charts

- A typical Structure chart is a top-down modular approach showing the functional hierarchy between the components. In addition, it also shows the *data* interfaces between the components.
- Structure chart is a graphical way to represent module decomposition hierarchy. They are dynamic models like the DFDs showing that show how one function calls the others.
- A structure chart is used as a design tool which supports in dividing and conquering a large software problem by breaking down a problem into modules that are small enough to be easily understood. The process is also called as top-down design or functional decomposition.

☞ Notations used in Structure Chart

A Structure chart shows the breakdown of a system to its lowest manageable levels arranged in a form of a tree.

- Each *module* is represented by a *rectangular box* containing the module name. The tree structure signifies the relationships between modules.
- The module *hierarchy* is displayed by linking rectangles with *lines*.
- *Inputs* and *outputs* are indicated with *arrows* - An arrow entering the box is its input and that is leaving the box is its output.
- *Data stores* are shown as *rounded rectangles* and user inputs as circles.

Example : Structure Chart for 'Paying Bill'

This structure chart represents data passing between modules. When the module 'Pay Purchase Bill' is executed, the pseudo code checks if the bill is already paid or not by searching for the payment receipt by executing 'Search Receipt' module and if the receipt is not found then it only it will execute the module 'Give Money to money Collector' and then the job is finished.

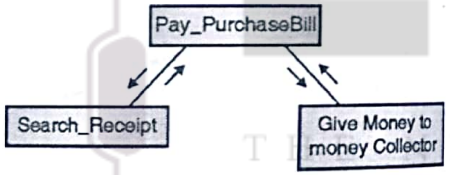


Fig. 5.12.1 : Structure Chart for 'Paying Bill'

☞ Steps to draw Structure Chart

Step 1: Identify the system processes: These processes are responsible for central processing functions that are not concerned with any input or output functions such as reading or writing data or data validation. Group such processes under a single function at the first-level in the structure chart.

Step 2: Identify input transformations: These are concerned with reading data or validating it and so on. Group such processes also under a single function at the first-level in the structure chart.

Step 3: Identify output transformations: These are concerned with preparing and formatting output and write it to the user's screen or other output devices.

☞ Structure chart is referred to as 'the master-plan' that signifies

- Size and complexity of the system number of easily identifiable functions and modules whether each identifiable function is a manageable module or should be broken down into smaller modules.

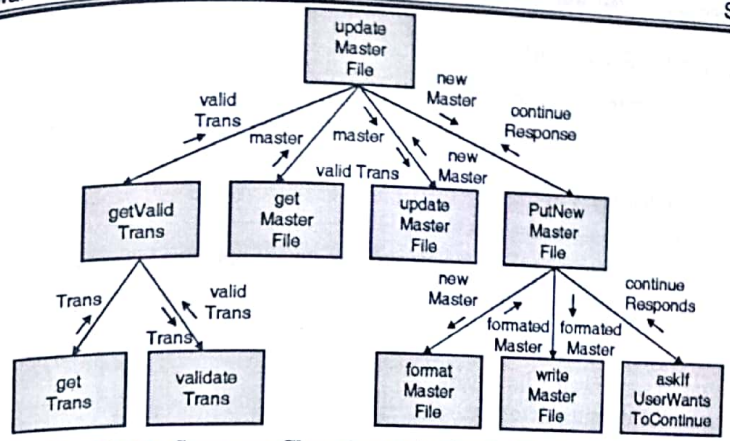


Fig. 5.12.2 : Structure Chart for 'Updating the Master File' function

5.13 Flow Charts

Flowchart is a modelling technique used in structured development and business modelling.

Flowchart is a graphical representation of the procedural logic of a computer program.

Example : Flowchart for 'Enrolment in University Course'.

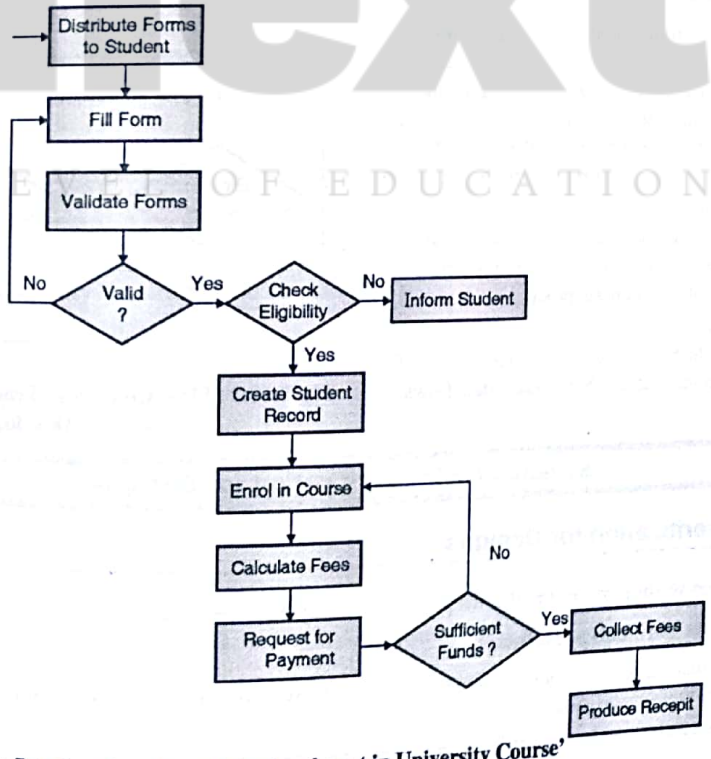


Fig. 5.13.1 : Flow Chart for 'Enrolment in University Course'

Notations used in Flow Chart

- *Rectangular boxes* represent the *activities* or tasks that are the computer instructions.
- *Diamonds* represent *decision points* i.e. a binary decision because there are two arrows leaving out from a decision.
- *Arrows* represent *flow of control.*
- Off-page connectors when diagrams get too big
- Input/output symbols represent printed reports and data storage options.

Example : Flowchart for 'Traditional Software Development'

Drawbacks

- Unless we take care, drawing flowcharts can be complicated and difficult to read.
- If the user's policy changes or if the systems analyst changes it several times before the user accepts it as correct, it will be time consuming and difficult job to redraw the flowchart each time.

Differences between Flowchart and the DFD

- DFDs show that all the bubbles (processes) in the system can be active simultaneously but the flow charts show individual process specification in a one-dimensional form i.e. the logic of the system flows uniformly from top to bottom.
- DFDs describe data flow within a system whereas, flow charts describe the detailed logic of a business process or business policy.
- Flowcharts represent sequencing of operations and DFDs represent data flows.

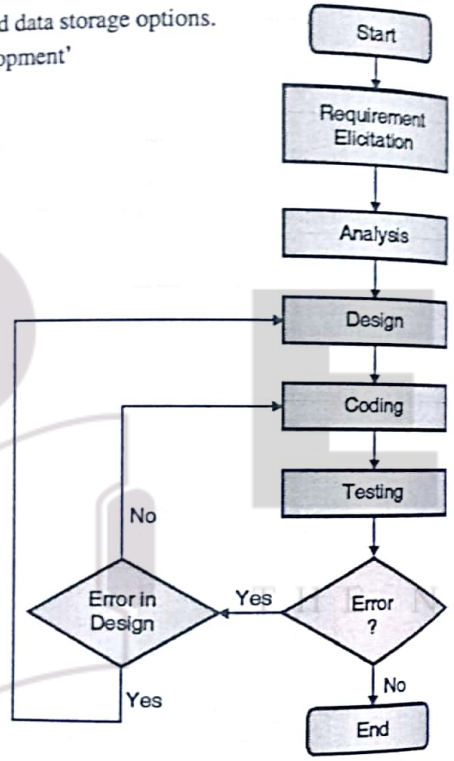


Fig. 5.13.2 : Flow Chart for 'Traditional Software Development'

Syllabus Topic : Verification for Designs

5.14 Verification for Designs

Design verification ensures that the product i.e. designed is the same as the product i.e. intended.

Design Verification Methods

1. **Demonstration :** It is done in actual or simulated environments. The cost of demonstration varies as per the complexity of the demonstration.
2. **Inspection :** It is done to verify the requirements related to physical characteristics.

3. **Analysis :** It is done to verify the design and is the most preferred method incase if testing is not feasible or incase when there is minimal risk.
4. **Testing :** It is the most expensive verification methods due to the project complexity as well as equipment and facility requirements. The most common method for validating this requirement is transportation testing.

Verification Activities

Fig. C5.5 : Verification Activities

1. Identification and preparation

- While developing a specification, the test engineer starts writing detailed test plan and procedures.
- Identifying the best approach of verification, identifying the measurement methods and required resources, tools and facilities.
- Once the verification plan is ready, it is reviewed by the design team to identify issues before finalizing the plan.

2. Planning

- Planning for verification is a simultaneous activity with core development activity which continues throughout the project life cycle. The verification plan is updated as and when any changes are made to design inputs.
- Planning includes the scope of the project, tools, test environment, development strategy.

3. Developing

- The test case development identifies a variety of test methods.
- Developing of design inputs which are unambiguous and verifiable.
- Using the output of one test as an input for subsequent tests.
- Traceability links are created to ensure that all the requirements are tested and the design output meets the design inputs.

4. Execution

- The test procedures created in development phase are executed as per the test plan.
- If actual doesn't match with the expected results, such issues are identified and logged as defect at this stage.
- Traceability matrix is created to ensure that all the requirements are tested and the design output meets the design inputs.

5. Reports

- The design verification report gives the detailed summary of verification results which includes the configuration management and test results and defects found during the verification activity.
- Design verification traceability report ensures that all the requirements have been tested and provided with appropriate results.

Syllabus Topic : Monitoring and Control for Design

5.15 Metrics, Monitoring and Control for Design

Design metrics focus on characteristics of program architecture with an emphasis on the architectural structure and the effectiveness of modules or components within the architecture.

Design Metrics

- **Size :** It is defined in terms of:
  - Population: it is the count of operations and classes.
  - Volume: It is the count of population at a given instant of time.
  - Length: It is the measure of a chain of interconnected design elements
  - Functionality: It defines the value delivered to the customer by an OO application.
- **Complexity**: It defines how classes or modules of a design are interrelated.
- **Volatility**: The design changes occur when requirements are modified or when modifications occur in other parts of an application resulting in adaptation of the design component. Volatility component measures the likelihood that a change will occur.
- **Coupling**: It is an indication of the relative interdependence among the modules.
  - The physical connections between the elements of the design like number of collaborations between sub systems, messages passed between objects; all this represent coupling within a system.
  - Coupling allows simple connectivity among modules which result in software that is easier to understand and less prone to errors that occur at one location and propagate throughout a system.
- **Cohesion** : It is the indication of the relative functional strength of a module.
  - A cohesive module performs a single task requiring little interaction with other components in other parts of a program.
  - Cohesion is doing only one thing.

Card and Glass define three software design measures in terms of complexity

1. *Structural complexity* is defined as the number of modules immediately subordinate to a module.
2. *Data complexity* provides an indication of the complexity in the internal interface for a module.
3. *System complexity* is defined as the sum of structural and data complexity.

Monitoring and Controlling Mechanisms

- This task lasts throughout the project and covers the development process
- Monitoring and Controlling is a reporting mechanism that reports about information flows and reviews.
- It controls the volatile changes
- Identifies and controls the risks
- Monitors all key parameters like cost, schedule, risks
- Takes corrective actions when needed

Review Questions

Q. 1 What is the need of system design?

Q. 2 What is the difference between Function Oriented and Object Oriented design.

Q. 3 State the differences between Coupling and Cohesion

Q. 4 What is meant by Abstraction and state its types.

Q. 5 What is the need of problem partitioning?

Q. 6 Draw structure chart and flow chart for 'order processing system'.

Q. 7 What activities are includes in design verification ?

Q. 8 Define the various methods of design verification.

Q. 9 What is the need of monitoring and controlling mechanism ?

Q. 10 State the difference between flowchart and DFD.

□□□

UNIT II

CHAPTER 6

Software Measurement and Metrics

Syllabus

Product Metrics – Measures, Metrics, and Indicators, Function-Based Metrics, Metrics for Object-Oriented Design, Operation-Oriented Metrics, User Interface Design Metrics, Metrics for Source Code, Halstead Metrics Applied to Testing, Metrics for Maintenance, Cyclomatic Complexity, Software Measurement - Size-Oriented, Function-Oriented Metrics, Metrics for Software Quality

Syllabus Topic : Product Metrics – Measures, Metrics and Indicators

6.1 Measures, Metrics and Indicators

Q. Explain the terms : measure, metric and indicator.

Measure

- A measure is a quantitative indication of the extent, capacity, or size of some attribute of a product or a process.

Metric

- A metric is the measurement of a particular characteristic of a program's performance or efficiency.
- A metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute.
- Metrics should be collected during and at the end of every phase in the SDLC. They are also a valuable input into process improvement.

Indicators

- An indicator is a combination of metrics that provides insight into the software process, project or product.

6.1.1 Need of Metrics

- Provides a means for control/status reporting
- Tracking Projects against plan - Provides a basis for estimation and facilitates planning for closure of the performance gap

- Identifies risk areas and take timely corrective actions
- Getting early warnings - Provides meters to flag actions for faster and more informed decision making
- Basis for setting benchmarks
- Basis for driving process improvements - Helps in identifying potential problems and areas of improvement
- Tracking process performance against business

6.1.2 Role of Software Metrics

- Software Development Life Cycle (SDLC) performs main role in case of creation of the software. There are primarily two things which are very important to measure at the time of software development which are ***quality*** of the software and ***progress*** of the software. Both things are measured by the SDLC.

Development or Management Team always focuses on the *quality and progress* parameters and for that purpose, it uses metrics for various purposes such as;

1. **Progress Checking :** As there is progress measure it is important to arrange some schedule for that. Actual Progress accessing is done at actual date as per mentioned in the schedule.
2. Making Quality software
3. **Costing and Scheduling :** are the important things which are concern with money and time respectively. Proper estimation of both make fruitful product with accuracy over time.

The Seven Core Metrics

Primarily there are seven core metrics which are classified on the basis of management issues and quality issues. Fig. 6.1.1 depicted metrics;

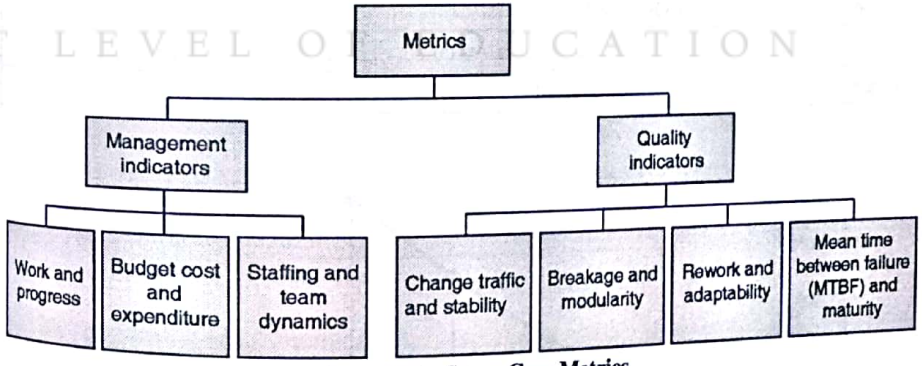


Fig. 6.1.1 : Seven Core Metrics

All the above metrics are classified into two different dimensions :

1. Static Value means objectives of the metrics
2. Dynamic trends concern with managing accomplishment of those objectives.

- All the above seven metrics are useful for managing the organization and project.
- Seven core metrics are based on the metric programs;
  - Common sense
  - Field experience

- Their Attribute
  - o They are very Simple to understand
  - o Each has definite objective
  - o Each time easy to collect, since automatic and nonintrusive collection is there
  - o Easy to interpret
  - o Hard to misinterpret
  - o Continuous assessment
  - o They are acts as communicating bridge between management and engineering personnel for the continuous progress and quality parameters.

I. Management Indicators

There are three important management parameters on which every manager can works for :

o Technical progress o Financial assets o Staffing progress

Project costing and scheduling is the critical job for most of the managers. At the time of the scheduling the project work assessing is again problematic for the managers.

A. Work and Progress

When any project starts then initial duty of the manager is scheduling the project. Planning is the most important task in the scheduling of iterative development of the project. The fact is when some things happen with plan then it is time bounded. Here the progress is most important fact which is carried out by the time scheduling.

The routine task allocated for this metrics for various teams is as follows :

Software teams	Duties
Architecture team	Use case demonstrated
Development team	SCOs closed
Assessment team	SCOs opened, test time executed, evaluation criteria congregate
Management team	Milestone targeted

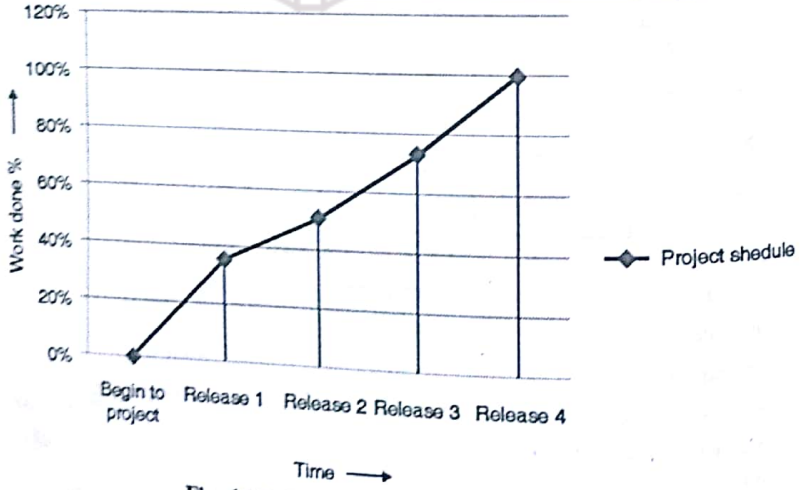


Fig. 6.1.2 : Progress against schedule time

B. Budgeted Cost and Expenditures

Work as per schedule time is important expenditure on the project as per budget cost is also important. Financial progress tracking is another major management indicator which is carried out by the specific standard format approved by the organization. Earned value system is the approach which is used to detailed information of cost and scheduling. All the parameters of the earn value system are indicated in the form of dollar. Some parameters mention below;

1. **Expenditure Plan :** The financial plan schedule spends for the project activity over the previous planned schedule is nothing but expenditure plan.
2. **Actual Progress :** Technically completed plan over the planned progress is actual progress.
3. **Actual Cost :** Whatever actually spent cost for the project profile over the actual schedule.
4. **Earned Value :** Planned cost of the actual progress.
5. **Cost Variance (CV)** = Actual cost – Earned value
   - Positive CV indicates – Over Budget
   - Negative CV indicates – Under Budget
6. **Scheduled Variance (SV)** = Planned cost – Earned Value
   - Positive SV indicates – Behind schedule situation
   - Negative SV indicates – ahead schedule situation

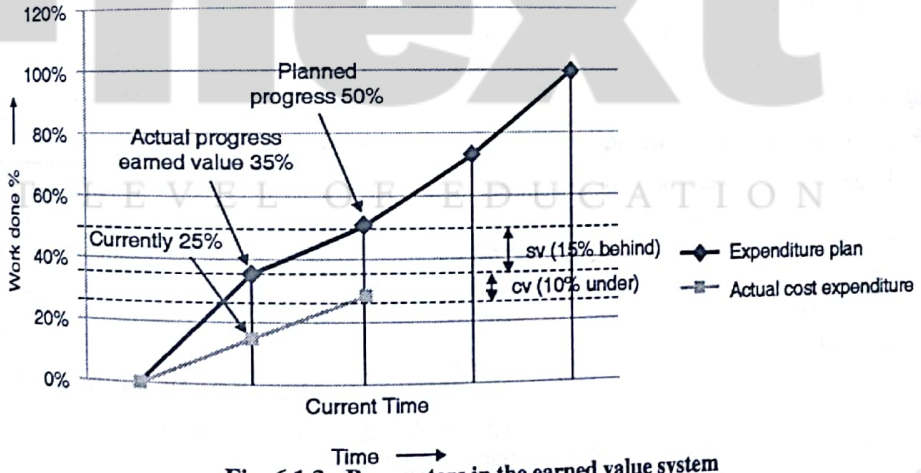


Fig. 6.1.3 : Parameters in the earned value system

C. Staffing and Team Dynamics

Quality staff is essential factor for developing quality software. When quality staff comes together and forms a dynamic team, they produce a long lived and continuously improving software product. It tracks the actual staffing against the planned staffing.

If new staff increases then work progress becomes slow. It affects on the overall project speed. New staff can take the productive time of exiting staff for speed. Low attrition of quality staff is the sign of the success. As the staff like software engineer is continuously working in the same company then they make good returns for the company by their experience.

Now days this kind of staff migrates from one place to another place because of the high and attractive salary package. In this case motivation to such staff is essential.

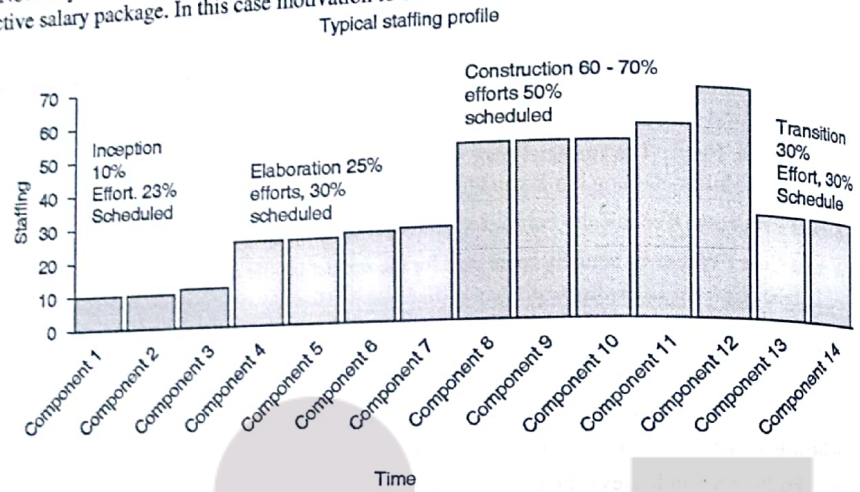


Fig. 6.1.4 : Typical Staffing Profile

II. Quality Indicators

There are four quality indicators which stand on the measurement of the change across evolving baseline engineering data.

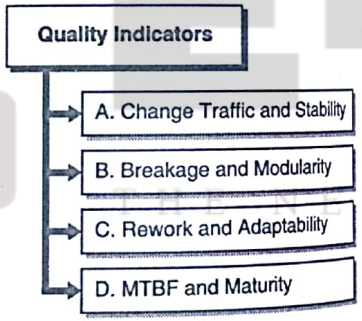


Fig. C6.1 : Quality Indicators

➜ A. Change Traffic and Stability

Progress and quality parameters of the software measure by the change traffic indicator.

1. **Change Traffic :** The number of the SCO's open and closed over the whole life cycle is known as ***Change Traffic***.
2. **Stability :** Relationship between open and closed SCO's is known as ***Stability***.

The information relates to the metrics is collected through overall change, its release type, all number of release, by team, by all its components, by all its subsystems, and so on.

➜ B. Breakage and Modularity

1. **Breakage :** The average extents of change, which is the amount of software baseline that needs rework is known as ***Breakage***.
2. **Modularity :** The average breakage trend over time is known as Modularity.

As time increase breakage trend is also increase, this indicates that product maintainability is suspect.

➜ C. Rework and Adaptability

1. **Rework :** The average cost of change, which is the effort to analyze, resolve and reset all changes to the software baseline is known as ***rework***.

2. **Adaptability :** The rework trend over time is known as adaptability.

➜ D. MTBF and Maturity

1. **Mean time Between Failures :** The average usage time between software faults is known as ***Mean Time Between Failures***. It is calculated by dividing the test hours by the number of type 0 -1 SCOs.
2. **Maturity :** The MTBF trend over time is known as Maturity.

As an error comes in the software projects then that are revised and prevent from those errors. Such errors are categorized into two types: Deterministic errors and non deterministic errors.

Table 6.1.1 show the differentiation table of deterministic and non- deterministic errors;

Table 6.1.1 : Difference between deterministic and non deterministic errors

Sr. No.	Deterministic Errors	Non - deterministic Errors
1.	Physicists named it as Bohr – bugs.	Physicists names it as Heisen – bugs.
2.	Class of errors that always result when the s/w is simulated in certain way.	Probabilistic occurrence of given situation.
3.	Reasons of occurs – coding and change in the single component.	Reasons of occurs – design time errors.

6.1.3 Types of Testing Metrics

Q. List various types of metrics.

Metrics can be categorized into four types :

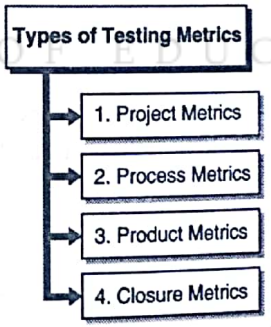


Fig. C6.2 : Types of Testing Metrics

➜ 6.2 Project Metrics

Project metrics are used by a project manager and software team to adapt project work flow and technical activities.

These metrics are useful in :

- minimizing the development schedule by making the necessary adjustment to avoid delays and mitigate problems.

- Assessing the product quality on an ongoing basis

1. Test coverage = (# of Test executed / # of Tests estimated) * 100
2. Defect Density = (Total number of defects found in all the phases + Post delivery defects)/Size

Defect arrival rate - This metric indicates the quality of the application/product under test.

Defect arrival rate = # Of Defects * 100 / # of Test Cases planned for Execution

6.3 Process Metrics

Process metric focuses on the quality achieved as a consequence of a repeatable or managed process. It is usually strategic and planned for long term.

1. Cost of Quality

☞ **% Cost of Quality = (green money + blue money + red money)*100/(Total efforts spent on project)**

where,

- Prevention Cost: (Green Money) : Cost of time spent by the team in implementing the preventive actions identified from project start date to till date.
- Appraisal Cost: (Blue Money): Cost of time spent on review and testing activities from the project start date to till date
- Failure Cost: (Red Money) : Cost of time taken to fix the pre and post-delivery defects. Customer does not pay for this

1. Delivered Defect Rate
2. Defect Injection Rate
3. Rejection Index - measures the Quality of the defects raised
4. Rejection Index = # of Defects rejected / # of Defects raised
5. Review Effectiveness = 100 * Total no. of defects found in review / Total no. of defects
6. Defect Removal Efficiency (DRE) = (No. of pre-delivery defects / Total No. of Defects) * 100

6.4 Product Metrics

Product metrics focus on the quality of deliverables. Product metrics are combined across several projects to produce process metrics

1. Test Case Design Productivity = # Of Test cases (scripts) designed/ Total Test case design effort in hours
2. Test Case Execution Productivity = # Of Test cases executed/ Total Test case executed effort in hours

6.5 Closure Metrics

1. Test Design Review Effort = (Effort spent on Test Case design reviews / Total effort spent on Test Case design) * 100

2. Test Design Rework Effort = (Effort spent on Test Case design review rework / Total effort for spent on Test Case design) * 100
3. KM Effort = (Total Effort spent on preparation of the KM artifacts / Total actual effort for the project) * 100

Syllabus Topic : Function-based Metrics

6.6 Function-based Metrics

Q. Explain Function point metric with an example.

The function point metric (FP) is a means for measuring the functionality delivered by a system.

Function point **data is used in two ways:**

1. as an estimation variable that is used to "size" each element of the software
2. as baseline metric collected from past projects

Baseline metric along with Estimation variable is used to develop cost and effort projections.

The basic units of the Universal Function Points (UFP) are :

- user inputs – user outputs – user inquiries – files
- external attributes i.e. the machine readable interfaces that are used to transmit information to another system.

Table 6.6.1 : Computing the UFP

Measuring Parameter	Count	Weighting Factor Simple	Average	Complex	
No. of user inputs		* 3	4	6	=
No. of user outputs		* 4	5	7	=
No. of user inquiries		* 3	4	6	=
No. of files		* 7	10	15	=
No. of external interfaces		* 5	7	10	=
CountTotal					

To compute UFP, the following formula is used;

$$FP = CountTotal * [0.65 + 0.01 * \sum (Fi)]$$

where, countTotal is the sum of all FP entries shown in the Table 6.6.1 and Fi (where i=1 to 14) are the 'complexity adjustment values' based on answers to the following question;

1. Does the system require reliable backup and recovery?
2. Are data communications required?

3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transactions to be built over multiple operations?
8. Is the master file updated on-line?
9. Are the inputs, outputs, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code reusable?
12. Are installation details included in the design?
13. Is the system designed to adjust on various installations of different organizations?
14. Is the application designed to incorporate the changes and is it easy of use ?

LOC (Lines of Code) & Estimation

- These metrics are useful estimators when a solution is formulated and programming language is known.
- FP and LOC are the accurate predictors of software development effort and cost.
- The relation between LOC and FP depends upon the programming language that is used to implement the software and the quality of the design.
- The Table 6.6.2 provides rough estimates of the average number of LOC required to build a FP in various programming languages.

Table 6.6.2 : Various Comparisons of FPs to LOC based upon the programming language

Language	LOC per UFP
Assembly	320
C	128
Fortran 77	105
Cobol 85	91
Ada 83	71
C++	56
Ada 95	55
Java	55
Visual Basic	35

Advantages

- The above data indicates that one LOC of C++ provides approximately 2.3 times the 'functionality' as one LOC of C.
- LOC and FP measures are used to derive productivity metrics.
- Use of higher level programming languages reduces the size, thus, the 'level of abstraction' also changes allowing more focus on architecture.

- The reduced size makes it easier to understand, reuse, maintain and import the packages of classes and objects.
- But, these higher-level abstractions often use high storages and communication bandwidths.

Example 6.6.1 :

Infosoft solutions Pvt. Ltd. uses the biometric system for keeping the attendance of the employees. This company takes the annual maintenance contract for their clients. The system is placed in this company's office that takes the details of the employee's database from the biometric. In the employees database system the employee id is stored, which is captured from biometric system. When the employee's thumb is scanned then the employee id is shown along with system data and time, which is also stored in database. The name of the employee, the time for which the employee has worked at client's locations along with the task they carried out at client's locations is even maintained in the database. The data which is generated from the same is taken into excel sheet by the staff from the office. The weekly report is generated by the staff where how many hours, the employee has worked is displayed. Even the monthly report is generated for the same. The number of hours in a week and average number of hours in a month for an employee is calculated. Explain at least 2 functional requirements given in this case study for each of the parameters required to calculate function point. Calculate unadjusted function point (UFP) assuming that the complexity level is "Simple" for this case study by taking the help of below given table.

Function point	Simple	Average	Complex
External input	3	4	6
External output	4	5	7
Logical internal file	7	10	15
External interface file	5	7	10
External inquiry	3	4	6

Solution :

Functional Requirements

1. **Enrol employee details :** Capture their fingerprints using image acquisition device such as the sensors and process them into usable form.
2. **Store images in database :** Distinctive measures of the enrolling images are extracted and stored in the database as a "template".
3. **Identifying and verifying those images** : These images are matched with the information by comparing a specific template with the templates already stored in the database so as to find if there is any match.
4. **Record the details of the matching identification otherwise generate an alert :** Record the entry date/time, employee id, exit date/ time of the matching identifications.
5. **Calculate the weekly worked hours / monthly worked hours** of every employee based on the recorded details where each record is identified upon the employee's fingerprints as an identification mark.

Calculating Unadjusted Function Point (UFP) for 'simple' complexity level from the given data :

The UFP calculation is broken into two categories with five sub-categories:

- Data Functions :
  - Internal Logical Files
  - External Interface Files
- Transactional Functions :
  - External Outputs
  - External Inputs
  - External Inquiries

Add Employee
Add Employee fingerprints
Scan thumb
Store date and time
} External Inputs

Generate employee details
Show attendance data and time
Show working hours
Show employee tasks
Generate excel sheet
Generate weekly/monthly report
} External Outputs

Query employee
Query client locations
Query tasks
Query over time hours
} External Inquiries

Function type	Number	Weight Factors Simple			
External Input (EI)	4	×	3	=	12
External Output (EO)	6	×	4	=	24
External Queries (EQ)	4	×	3	=	12
Internal Datasets (ILF)	5	×	7	=	35
Interface (EIF)	3	×	5	=	15
	Unadjusted function points (UFP)			=	98

Syllabus Topic : Metrics for Object-Oriented Design

6.7 Metrics for Object-Oriented Design

- OO methodology supports continuous integration of subsystems, classes, interfaces thus, increasing the chances of early detection of risks and incremental corrections without hampering the stability of the development process.

- OO methodology allows 'Architecture first approach' in which integration is an early and continuous life-cycle activity that brings stability in development, allows development and configuration of components in parallel.
- OO architecture creates a clear separation between the unrelated elements of a system, and includes firewalls that prevent a change in one part of the system that may be caused due to the errors in other part of the system thus, rending the structure of the entire architecture.
- OO metrics are used to evaluate the quality of the software. The software engineering metrics are usually associated with coupling, cohesion, reliability, maintainability and fault-proneness. But the OO metrics are specially concerned with;

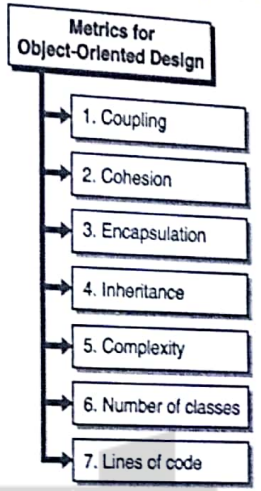


Fig. C6.3 : Metrics for Object-Oriented Design

1. Coupling

It refers to the association between modules. Coupling is a measure of interdependence of two objects.

Example : Objects X and Y are said to be coupled if a method of X calls or accesses a method or variable of Y.

$$\text{Coupling Factor (CF)} = \frac{\text{number of non-inheritance couplings}}{\text{maximum number of inheritance and non-inheritance couplings in a system}}$$

where, , Inheritance couplings arise from derived classes, inherited methods and attributes form its base class.

Desirable value of CF is lower.

2. Cohesion

It refers to how closely the operations in a class are related to each other. There are two ways of measuring cohesion :

i. Calculate the percentage of methods in a class that used the data field - Average the percentages then subtract from 100%. Lower percentages mean greater cohesion of data and methods in the class.

ii. Count the number of disjoint sets produced from the intersection of the sets of attributes used by the methods.

Desirable value of Cohesion is higher - High cohesion increases complexity, thereby increasing the likelihood of errors

3. Encapsulation

It refers to hiding of information means all that is seen of an object is its interface. There are two types of encapsulation measures.

i. **Attribute Hiding Factor (AHF) :** It measures the percentage of invisibilities of attributes in classes i.e. An attribute is said to be visible if it can be accessed by another class or an object. Attributes should be "hidden" within a class by declaring them as private. Desirable value of AHF is higher.

$$AHF = \frac{\text{sum of the invisibilities of all attributes defined in all classes}}{\text{total number of attributes defined in the project}}$$

ii. **Method Hiding Factor (MHF) :** It measures the percentage of invisibilities of methods in classes i.e. the percentage of total classes from which the method is not visible. Desirable value of MHF is higher.

$$MHF = \frac{\text{sum of the invisibilities of all methods defined in all classes}}{\text{total number of methods defined in the project}}$$

➜ **4. Inheritance**

It decreases the complexity by reducing the number of operations and operators, but on the contrary, it makes the maintenance and design difficult. There are two metrics to measure the amount of inheritance in terms of depth and breadth of the inheritance hierarchy.

i. **Depth of Inheritance Tree (DIT) : :** It is the depth from the current class node to the parent class node in the hierarchy tree and is measured by the number of ancestor classes The classes involved in multiple inheritance have maximum DIT. The deeper the class is within the hierarchy, the greater is the potentiality of reusing the inherited methods but increases the complexity. Desirable value of DIT is low.

ii. **Number of Children :** It measures the number of direct subclasses for each class. A class with large number of children is difficult to modify and test. Desirable value of NOC is low.

➜ **5. Complexity**

A class with more functions than its parent class is considered to be more complex and more error prone thereby limiting the possibility of reuse.

➜ **6. Number of Classes**

The projects having more number of classes are better abstracted. Desirable value of Number of Classes is higher.

➜ **7. Lines of Code**

The projects with fewer lines of code have superior design and require less maintenance. Desirable value of LOC is lower.

Syllabus Topic : Operation-Oriented Metrics

6.7.1 Operation-oriented Metrics

Q. What is the need of OO metrics and explain in brief.

- Average Operation Size (OSavg) = # of messages sent by the operation
- Operation Complexity (OC)
- Average number of Parameters per Operation (NPavg)

6.7.2 Use Case Point (UCP) Estimation Method

The UCP estimation method was introduced by Gustav Karner.

This method is implemented using a spread sheet (excel sheet).

- Each actor and use case is categorized according to complexity and is assigned a weight.
- *Complexity* of a use case is measured in number of transactions.
- The *unadjusted use case points* are calculated by adding weights for each actor and use case based on *13 technical factors* and *8 environmental factors*.

13 Technical Factors		
Sr. No.	**Factor description**	**Weight**
1.	Distributed system	2
2.	Response or throughput performance objective	1
3.	End-user efficiency	1
4.	Complex internal processing	1
5.	Code must be reusable	1
6.	Easy to install	0.5
7.	Easy to use	0.5
8.	Portable	2
9.	Easy to change	1
10.	Concurrent	1
11.	Includes special security features	1
12.	Provides direct access for third parties	1
13.	Special user training facilities are required	1
8 Environmental Factors		
Sr. No.	**Factor description**	**Weight**
1.	Familiar with RUP	1.5
2.	Application experience	0.5
3.	Object-oriented experience	1
4.	Lead analyst capability	0.5
5.	Motivation	1
6.	Stable requirements	2
7.	Part-time workers	−1
8.	Difficult programming language	−1

- The unadjusted use case weights (UUCW) by adding weights for each use case.
- The unadjusted actor weight (UAW) is calculated by adding weights for each actor.
- The unadjusted use case points (UUCP) is calculated as :

$$UUCP = UAW + UUCW.$$

- Calculate technical factor, $TF = .6 + (.01 * \Sigma 1..13Tn * Weightn)$.
- Calculate environmental factor,
- $EF = 1.4 + (-.03 * \Sigma 1..8Fn * Weightn)$.
- Calculate Use Case Points as: $UCP = UUCP * TF * EF$
- Finally, calculate UCP * Productivity factor

Syllabus Topic : Other Testing Metrics

6.8 Other Testing Metrics

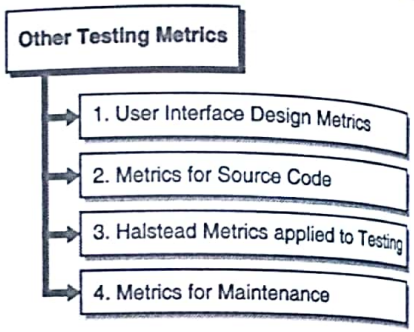


Fig. C6.4 : Other Testing Metrics

➜ **1. User Interface Design Metrics**

Layout appropriateness: It is a function of layout entities, the geographic position and the "cost" of making transitions among entities.

➜ **2. Metrics for Source Code**

Halstead's Software Science is a collection of all metrics predicated based on the number of operators and operands within a component.

➜ **3. Halstead Metrics applied to Testing**

Following design metrics have a direct influence on the "testability" of an OO system.

- Lack of cohesion in methods (LCOM).
- Public access to data members (PAD).
- Fan-in (FIN).
- Percent public and protected (PAP).
- Number of root classes (NOR).
- Number of children (NOC) and depth of the inheritance tree (DIT).

➜ **4. Metrics for Maintenance**

Q. Brief about metric of maintenance.

- Software Maturity Index (SMI) = $[M_T - (F_a + F_c + F_d)] / M_T$

where,

M_T : Total # of modules in the current release

F_a : # of modules added in the current release

F_c : # of modules changed in the current release

F_d : # of modules deleted in the current release

Syllabus Topic : Cyclomatic Complexity

6.9 Cyclomatic Complexity

- Cyclomatic Complexity testing is a process that measures the complexity of a program.
- Steps to calculate the Cyclomatic Complexity are as follows:

Step 1 : A program structure is first represented through a flowchart. It is a traditional means for pictorially describing a program's logic because describing the program code from a detailed flow chart is a very simple process.

Step 2: Flow charts are then converted in to a control flow graph containing only the nodes and edges. *Steps to convert the flow chart into flow graph are listed down:*

Step 2.1 : Identify the predicates (decision points) in the flow chart.

Step 2.2 : Ensure that the predicates are simple else break up a condition into simple predicates.

Step 2.3 : If a set of sequential statements is followed by a simple predicate, combine all the sequential statements and the predicate into one node.

Step 2.4 : If a set of sequential statements is followed by a simple predicate, combine all the sequential statements and the predicate check into one node and have two edges emanating from this one node. Such nodes with two edges emanating from them are called predicate nodes.

Step 2.5 : Make sure that all the edges terminate at some same node. Add an extra node at the end to represent that all the sets of sequential statements reach to that node.

Step 3: Calculate Cyclomatic Complexity (CC) using one of the either two methods:

Method 1 : From flow Chart, CC = P + 1 where P is number of predicates.

Method 2 : From Flow Graph, CC = E-N+2 where E is no. of edges, N is no. of nodes.

Method 3 : From Flow Graph, CC = # of closed Regions + 1 open Region

Example 6.9.1 :

1.	Wait for Card to be inserted	2.	IF card is valid THEN
3.	display "Enter PIN number"	4.	IF PIN is valid THEN
5.	select transaction	6.	ELSE
7.	display "PIN invalid"	8.	ELSE
9.	Reject card	10.	END

Solution :

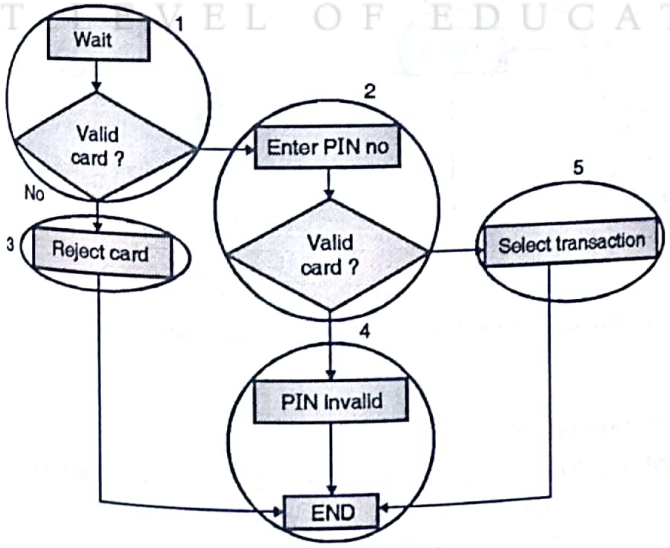


Fig. P. 6.9.1

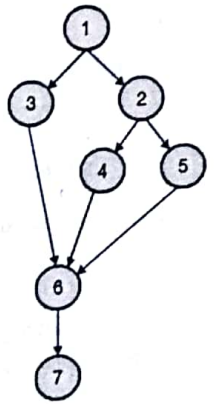


Fig. P . 6.9.1(a)

Number of Conditions (predicates) to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 Open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 6 7

Path 2: 1 2 4 6 7

Path 3: 1 2 5 6 7

Example 6.9.2 :

Program with complex predicates	Same Program with simple predicates

Read (A)
If A > 0 and A < 5 Then
Print "A"
End If
	Read (A)
	If A > 0 Then
	If A < 5 Then
	Print "A"
	End If
	End If

Solution :

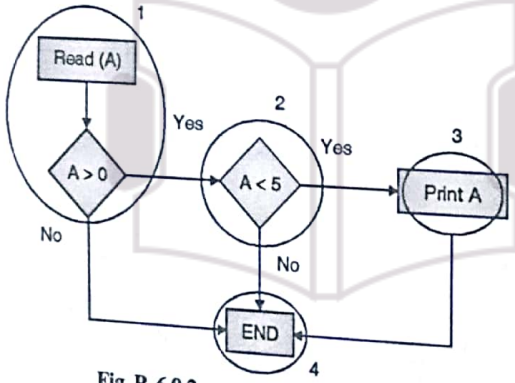


Fig. P. 6.9.2

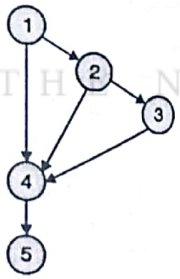


Fig. P. 6.9.2(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 Open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 4 5

Path 2: 1 2 4 5

Path 3: 1 2 3 4 5

Example 6.9.3 :

```
1.  Read A                  2.  Read B                  3.  IF A > 0 THEN
4.  IF B = 0 THEN           5.  Print "No values"       6.  ELSE
7.  Print B                 8.  IF A > 21 THEN          9.  Print A
10. ENDIF                   11. ENDIF                   12. ENDIF
```

Solution :

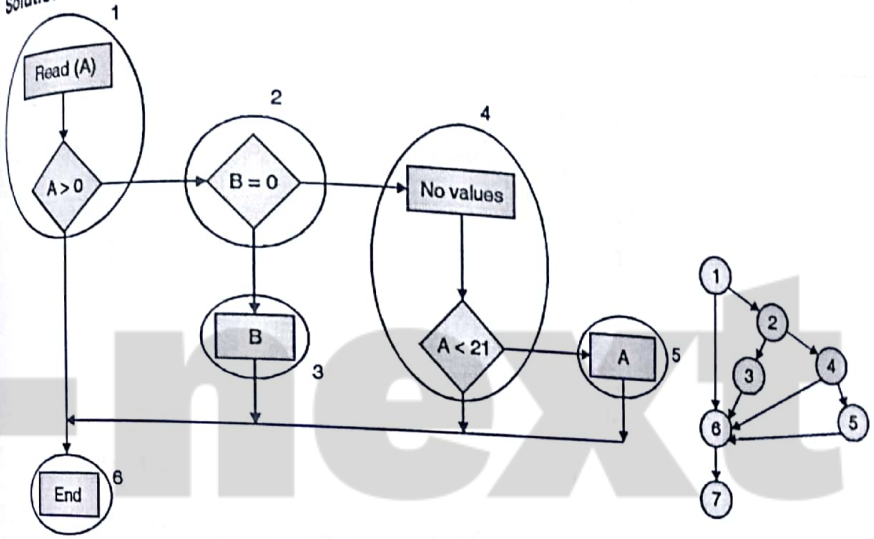


Fig. P. 6.9.3

Fig. P. 6.9.3(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 3

CC = P + 1 = 3 + 1 = 4

CC = # of Closed Regions + 1 open Region = 3 + 1 = 4

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 6 7 Path 2: 1 2 3 6 7 Path 3: 1 2 4 6 7

Path 4; 1 2 4 5 6 7

Example 6.9.4 :

```
1.  Read A                  2.  Read B                  3.  IF A < 0 THEN
4.  Print "A negative"      5.  ELSE                    6.  Print "A positive"
7.  ENDIF                   8.  IF B < 0 THEN           9.  Print "B negative"
10. ELSE                    11. Print "B positive"      12. ENDIF
```

Solution :

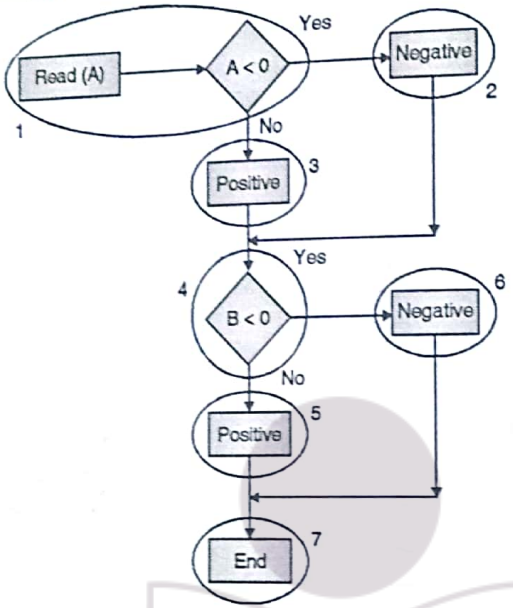


Fig. P. 6.9.4

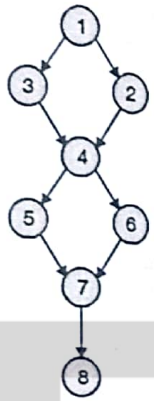


Fig. P. 6.9.4(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 4 5 7 8 Path 2: 1 3 4 6 7 8 Path 3: 1 2 4 5 7 8

Path 4; 1 2 4 6 7 8

Example 6.9.5 :

Check the cyclomateric complexity for a program of adding 100 integers. It should also check for valid boundaries and valid values. Design the Test Cases for such code.

Solution :

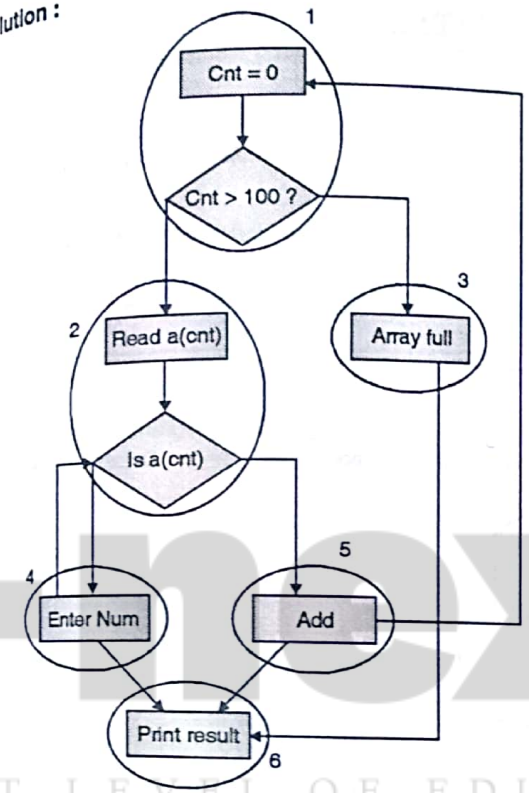


Fig. P. 6.9.5

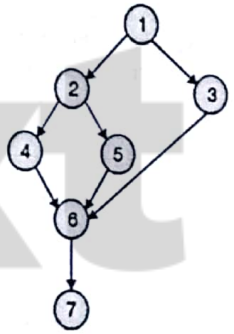


Fig. P. 6.9.5(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 6 7

Path 2: 1 2 4 6 7

Path 3: 1 2 5 6 7

Test Description :

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
No. of values in array must be 100 integers to perform addition	= 100 numbers	< 100 numbers > 100 numbers Non digit	100 numbers	99 numbers 101 numbers

Test case :

Test Case Id	Conditions	Input	Expected Output
1	No. of values in array must be 100 integers to perform addition	Values till count =100	Addition of Numbers
2		Values less than count =100	Accept more numbers
3		Values greater than count =100	Array Full Warning
4	Non digits not acceptable	Any character or symbol	Enter number warning

Example 6.9.6 :

A program reads three integer values as three sides of a triangle. The program prints a message indicating that whether the triangle is right angle triangle or equilateral triangle. Draw the flow graph, calculate cyclometric complexity.

Solution :

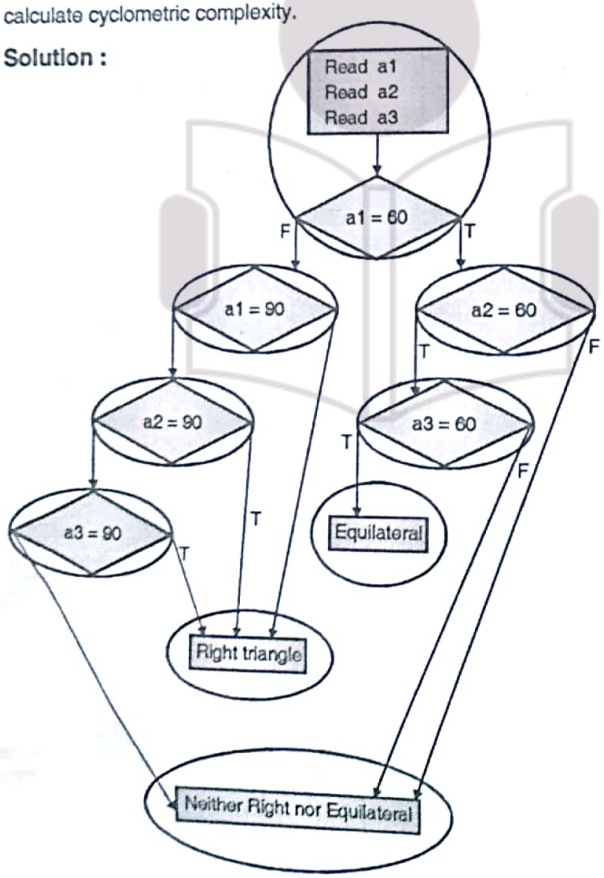


Fig. P. 6.9.6

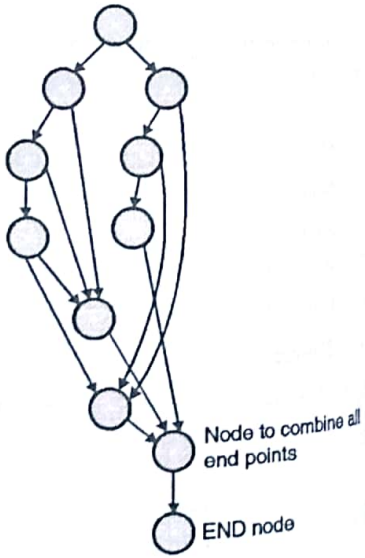


Fig. P. 6.9.6(a)

$CC = P + 1 = 6 + 1 = 7$

$CC = \#$ of Closed Regions + 1 open Region $= 6 + 1 = 7$

Example 6.9.7 :

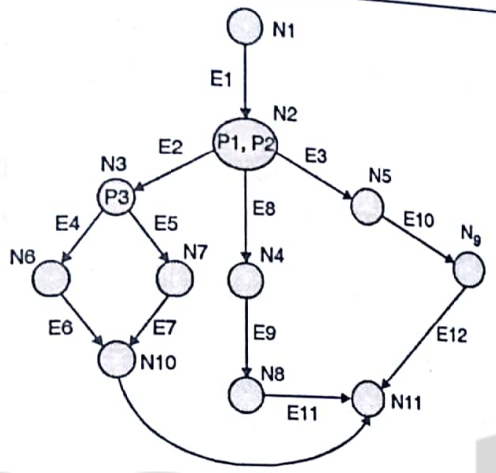


Fig. P. 6.9.7

Solution :

$CC = P + 1 = 3 + 1 = 4$

$CC = E - N + 2 = 13 - 11 + 2 = 2 + 2 = 4$

Syllabus Topic : Software Measurement

6.10 Software Measurement

Q. Based on which two parameters, the software is measured. Explain.

Organizations combine the metrics that come from different individuals or projects depending on the size and complexity.

1. Size oriented metrics (lines of code approach) such as errors per KLOC (thousand lines of code), defects per KLOC, documentation per KLOC, errors per person-month.
2. Function oriented metrics (function pointed approach) such as errors per FP, defects per FP, documentation per FP, FP per person-month

Syllabus Topic : Metrics for Software Quality

6.11 Metrics for Software Quality

Q. Explain metrics of software Quality.

The following are the software quality factors and the needed metrics and measurements for each :

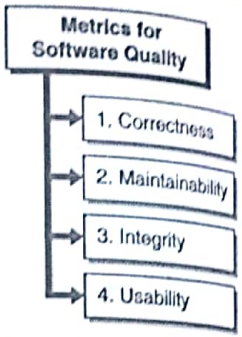


Fig. C6.5 : Metrics for Software Quality

→ **1. Correctness**

- Correctness is measured as the degree to which the software performs its required functionality.
- The most common measure for correctness is number of defects per KLOC (Kilo Lines Of Code) where a defect is defined as a verified lack of conformance to requirements.

→ **2. Maintainability**

- Maintainability is the ease with which a program can be :
- An application can be corrected if an error is encountered
- a defect can be fixed
- a application is adapted in a changed environment
  - a application is enhanced.
- A simple time oriented metric is Mean Time To Change (MTTC) i.e. the time it takes to analyze the changed request, design an appropriate modification, implement the changed modification, test it and distribute the change for all users. Below is the figure which gives the scenario of online detection and offline repair. The measures MTBF (Mean Time Between Failure), MTTD (Mean Time To Detect) and MTTR (Mean Time To Repair) are the average times to failure, to detection and to repair.

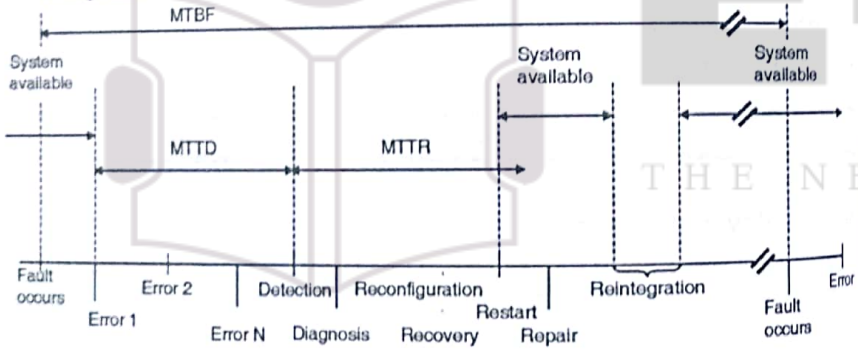


Fig. 6.11.1 : MTBF, MTTD and MTTR measures of maintainability

→ **3. Integrity**

- This attribute measures a system's ability to withstand attacks to its security. Attacks can be made on all three components of software: programs, data and documents.
- We cam measure integrity through following two additional attribute :
  1. Threat: It is the probability that an attack of a specific type will occur within a given time.
  2. Security: It is the probability that the attack of a specific type will be repelled. The integrity of a system van be then defined as :

  Integrity = Σ (1-(threat * (1-security)))

→ **4. Usability**

- If a program is not easy to use, it is often meant to failure. Even if the functions that it performs are valuable.

- Usability is an attempt to quantify ease-of-use and can be measured in terms of characteristics.

Defect Removal Efficiency

Q. What is Defect Removal Efficiency ?

- It is a quality metric that provides benefit at both the project and process level.
- DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.
- DRE is defined as: DRE = E/(E+D) where E is the number of errors found before delivery of the software to the end user and D is the number of defects found after delivery. The ideal value for DRE is 1 i.e. no defects are found in the software. Generally, D is greater than 0 but DRE can be 1 when, as E increases the overall value of DRE begins to approach 1.

Defect Detection Index

Defect Detection Index = # of defects detected in each phase / total # of defects planned to be detected in each phase

Defect Amplification and Removal

- A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design and coding steps of the software engineering process.

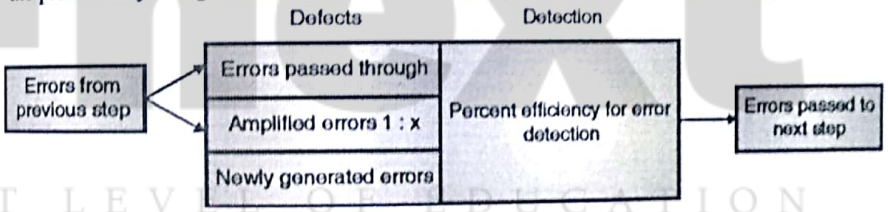


Fig. 6.11.2 : Defect Amplification model

- In case if the technical reviews fail to discover the newly generated defects, these may get amplified. The figure represents the percent of efficiency in detecting errors

Establishing a Software Metric Program

Following are the steps to conduct a goal-driven software metric program :

Step 1 : Identify the business goals
Step 2 : Identify what you want to learn
Step 3 : Identify the sub goals
Step 4 : Identify the related entities and attributes
Step 5 : Formalize your goals
Step 6 : Identify the quantifiable questions so as to achieve your goals.
Step 7 : Identify the data elements to help answer your questions.
Step 8 : Define the measures to be used
Step 9 : Identify the actions to implement the measures.
Step 10 : Prepare a plan for implementing the measures.

Review Questions

Q. 1 Explain the terms : measure, metric and indicator.

Q. 2 List various types of metrics.

Q. 3 What is Defect Removal Efficiency ?

Q. 4 Explain Function point metric with an example.

Q. 5 What is the need of OO metrics and explain in brief.

Q. 6 Brief about metric of maintenance.

Q. 7 Explain metrics of software Quality.

Q. 8 Based on which two parameters, the software is measured. Explain.

□□□

UNIT II

CHAPTER 7

Software Project Management Estimation

Syllabus :

Estimation in Project Planning Process – Software Scope and Feasibility, Resource Estimation, Empirical Estimation Models – COCOMO II, Estimation for Agile Development, The Make/Buy Decision

Syllabus Topic : Estimation in Project Planning Process

7.1 Estimation in Project Planning Process

- Planning and estimating are iterative processes which continue throughout the course of a project.
- Software costs often dominate computer system costs. The costs of software are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- The total cost required in developing a software product can be categorized as below :
  i. 60% of development costs.
  ii. 40% are testing costs.

Cost estimating problems occur due to following uncertainties

- Inability to accurately size a software project, Inability to accurately specify a software development and support environment.
- Improper assessment of staffing levels and skills, and Lack of well-defined requirements for the specific software activity being estimated.

Four types of cost estimates represent various levels of Reliability

- **Conceptual Estimate :** Often inaccurate because there are too many unknowns.
- **Preliminary Estimate :** Used to develop initial budget, more precise.
- **Detailed Estimate :** Serves as a basis for daily project control.
- **Definitive Estimate :** Accuracy should be within 10% of final cost.

Cost Estimations are constructed based on the following tasks

- Identifying the purpose and scope of the new system - New software development, software reuse, COTS integration, etc.
- Choosing an estimate type - Conceptual, preliminary, detailed, or definitive type estimates.
- Identifying system performance and/or technical goals.
- Laying out a program schedule.
- Collecting, evaluating, and verifying data.
- Choosing, applying, cross-checking estimating methods to develop the cost estimate.
- Performing risk and sensitivity analysis.
- Providing full documentation.

Syllabus Topic : Software Scope and Feasibility

7.1.1 Software Scope and Feasibility

- Software scope is defined as :
  - The functions and features that are to be delivered to end users
  - The input and output data of the system
  - The performance, constraints, interfaces, and reliability of the system
- Software Scope can be defined using two techniques :
  1. A narrative description
  2. A set of use cases
- Once the scope is decided, the feasibility is addressed
- Software feasibility has four dimensions
  1. **Technology :** Is the project technically feasible so as to reduce the defects.
  2. **Finance :** Is the project financially feasible i.e. can it be developed in the cost estimated by the software organization and its client.
  3. **Time :** Will the project be delivered in the estimated time.
  4. **Resources :** Does the software organization have all those resources required for successfully accomplishing the project?

Syllabus Topic : Resource Estimation

7.2 Resource Estimation

Software engineering resources can be categorized into three kinds :

1. People
   - Number of people required
   - Skills required in them
   - Geographical location from where they will work
2. Development environment
   - Software tools required in Project process

   - Computer hardware
   - Network resources
3. Reusable software components
   - Off-the-shelf components
     - Such components are either from a third party or available from a previous project
     - Such components are ready to use i.e. fully validated and documented
   - Full-experience components
     - Components are similar to the software that needs to be built
   - Partial-experience components
     - Components are somewhat related to the software that needs substantial modification to be built as required and also have substantial risks.
   - New components
     - Components must be built from scratch and can have a high degree of risk.

Each of the above resources is specified with :

- A description about the resource
- A statement of availability
- The time when the resource will be required
- The duration of time that the resource will be applied

7.2.1 Software Cost Estimation Process

Cost Estimation process comprises of 4 main steps :

Step 1 : Estimate the size of the development product

Size of the software may depend upon : lines of code, inputs, outputs, functions, transactions, features of the module and etc.

Step 2 : Estimate the effort in person-hours

The effort of various Project tasks expressed in person-hours is influenced by various factors such as :

- Experience/Capability of the Team members.
- Technical resources.
- Familiarity with the Development Tools and Technology Platform.

Step 3 : Estimate the schedule in calendar months

The Project Planners work closely with the Technical Leads, Project Manager and other stakeholders and create a Project schedule. Tight Schedules may impact the Cost needed to develop the Application.

Step 4 : Estimate the project cost in dollars (or other currency)

Based on the above information the project effort is expressed in dollars or any other currency.

7.3 Cost Estimation Techniques

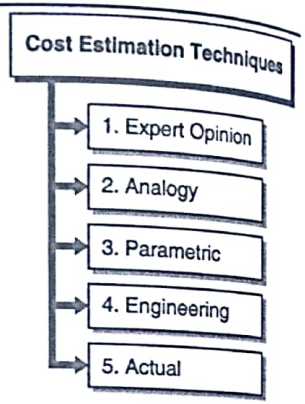


Fig. C7.1 : Cost Estimation Techniques

1. Expert Opinion

Also called as Delphi method, proposed by Dr. Barry Boehm is useful in assessing differences between past projects and new ones for which no historical precedent exists.

Advantages

- Little or no historical data needed.
- Suitable for new or unique projects.

Disadvantages

- Very subjective.
- Experts may do partiality
- Qualification of experts may be questioned.

2. Analogy

- Estimates costs by comparing proposed programs with similar, previously completed programs for which historical data is available.
- Actual costs of similar existing system are adjusted for complexity, technical, or physical differences to derive new cost estimates
- Analogies are used early in a program cycle when there is insufficient actual cost data to use as a detailed approach
- Compares similarities and differences
- Good choice for a new system that is derived from an existing subsystem.

Advantages

- Inexpensive – Easily changed – Based on actual experience (of the analogous system)

Disadvantages

- Very Subjective
- Large amount of uncertainty
- Truly similar projects must exist and can be hard to find
- Must have detailed technical knowledge of program and analogous system

3. Parametric :

Utilizes statistical techniques. Can be used prior to development.

Advantages

- Can be excellent predictors when implemented correctly
- Easily changed
- Objective
- Once created, CERs are fast and simple to use
- Useful early on in a program

Disadvantages

- Often lack of data on software intensive systems for statistically significant CER

- Does not provide access to subtle changes
- Top level; lower level may be not visible
- Need to be properly validated and relevant to system

4. Engineering :

Also referred to as ***bottoms up*** or detailed method.

- Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.
- Future costs for a system are predicted with a great deal of accuracy from historical costs of that system.
- Involves examining separate work segments in detail.
- Estimate is built up from the lowest level of system costs.
- Includes all components and functions.
- Can be used during development and production.

Advantages

- Objective – Reduced uncertainty

Disadvantages

- Expensive – Time Consuming – Not useful early on
- May leave out software integration efforts

5. Actual

- Decides future costs on recent historical costs of same system.
- Used later in development or production.
- Costs are calibrated to actual development or production productivity for your organization.

Advantages

- Most accurate – Most objective of the five methodologies

Disadvantages

- Data not available early
- Time consuming
- Labour intensive to collect all the data necessary

Choice of methodology depends upon

- **Type of system** - software, hardware, etc
- **Phase of program** - Development, Production, Support
- **Available data** - Historical data points from earlier system versions or similar system or Technical parameters of system.

7.4 Cost Estimation Parameters

- Various models (such as COCOMO, Co-star etc.) are available for estimating the cost of software development.

- All these cost estimating models can be represented on the basis of five basic parameters.

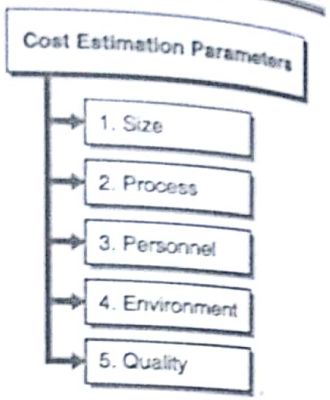


Fig. C7.2 : Cost Estimation Parameters

➔ 1. Size

The size of the proposed software product is weighed in terms of components i.e. ultimately in terms of the number of source code instructions or the number of functions required in developing the proposed product.

➔ 2. Process

The process includes the phases and activities carried out in each phase. So whatever process used to produce the proposed product is measured on the ability of the process to avoid unnecessary activities such as rework, bureaucratic delays, communications overhead and such other overhead activities which may delay the delivery of the product.

➔ 3. Personnel

This deals with the capabilities and experience of software engineering *personnel* (team members) in the field of computer science issues and the applications domain issues of the project. It also depends upon the personnel's familiarity with the Development Tools and Technology Platform.

➔ 4. Environment

The environment constitutes of the tools and techniques that are required to develop efficient software and also to automate the process.

➔ 5. Quality

- The required quality of the proposed product depends upon the features, performance, reliability, and adaptability of the software.
- The above described five parameters (size, process, personnel, environment and quality) can be related with each other so as to calculate the estimated cost for the proposed software development.

$$\text{Effort/Cost} = (\text{Personnel})(\text{Environment})(\text{Quality})(\text{Size}^{\text{process}})$$

7.5 Pragmatic Software Cost Estimation

Developers and customers always had an argument on software cost estimation models and tools.

Three most common topics of their argument

1. Selection of Cost Estimation Model

There are various cost estimation models (COCOMO, COSTXPERT, CHECKPOINT, SLIM, ESTIMACS, Knowledge Plan, SEER, SOFTCOST, Co-star, REVIC, Price-S, ProQMS etc.) that are based on statistically derived *cost* estimating relationships (CERs) and various estimating methodologies.

2. Whether to measure software size on the basis of lines of code (LOC) or function points (FP) ?
   - Most of the cost estimation models are bottom-up i.e. substantiating a target cost rather than *top-down* i.e. estimating the 'should' cost.
   - The Fig. 7.5.1 depicts predominant cost estimation practices where :

- First, project manager defines the target cost of the software.
- Later, he manipulates the parameters and does the sizing until the target cost is justified.
- The target cost is defined so as :
  - to win the proposal,
  - to solicit the customer funding,
  - to attain the internal corporate funding, or
  - to achieve some other goal.
- The process described in Fig. 7.5.1 forces the software project manager to check the risks associated in achieving the target costs and also discuss this problem with other stakeholders.

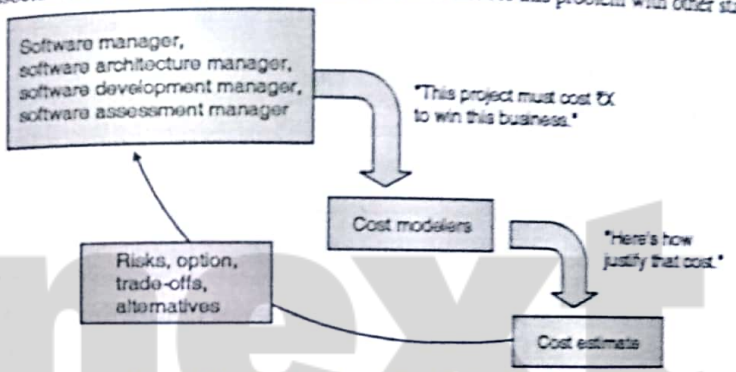


Fig. 7.5.1 : Predominant Cost Estimation Process

1. Factors that lead to good cost estimation

Good software cost estimation may be based on the following attributes :

- The cost is estimated collectively by the project manager, architectural team, development team, and test team.
- The estimated cost is acknowledged and supported by all stakeholders.
- The cost estimation is based on some well-defined model.
- The cost estimation is based on the databases of similar type of project experiences that use similar methodologies, similar technologies, and similar environment with similar type of stakeholders.
- The key risk areas are detected and accordingly the probability of success is determined.

Syllabus Topic : Empirical Estimation Models

7.6 Empirical Estimation Models

- Estimation models use empirically derived formulas such as LOC or FP to predict the required efforts.
- The computed values for LOC or FP are entered into an estimation model
- The empirical data used in deriving the LOC or FP for these models are derived from the sample projects.

7.6.1 COCOMO Model

- **Constructive Cost Model** (COCOMO) is one of the earliest cost models widely used by the cost estimating community.
- COCOMO was originally published in Software Engineering Economics by Dr. Barry Boehm in 1981.
- COCOMO is a regression-based model that considers various historical programs software size and multipliers.
- COCOMO's most fundamental calculation is the use of the Effort Equation to estimate the number of Person-Months required in developing a project.
- COCOMO stands for Constructive Cost Model. It is the oldest cost estimation model that is popularly used in the process of cost estimation.
- COCOMO model estimates the cost by considering the size and other quality aspects of the similar type of historical (previously developed) programs.
- COCOMO calculates Efforts i.e. it estimates the number of Person-Months required in developing a project.

Number of person months * loaded labour rate = Estimated Cost

- Most of the other estimates (requirements, maintenance, etc) are derived from this quantity.
- COCOMO requires as input the project's estimated size in Source Lines of Code (SLOC).
- Initial version published in 1981 was COCOMO-81 and then, through various instantiations came COCOMO2.
- COCOMO-81 was developed with the assumption that a waterfall process would be used and that all software would be developed from scratch.
- Since then, there have been many changes in software engineering practice and COCOMO2 is designed to accommodate different approaches to software development.

The COCOMO model is based on the relationships between the two formulate

Formulae 1 : Development effort is based on system size.

MM = a.KDSI b

where,

MM is the effort measured in Man per Moths

KDSI is the number of Source Instructions Delivered in a Kilo (thousands).

Formulae 2 : Effort (MM) and DEVelopment Time.

TDEV = c.MM d

where,

TDEV is the development time.

In both the above formulas, we have used the coefficients a, b, c and d which are dependent upon the 'mode of development'.

According to Boehm, the mode of development can be classified into following 3 distinct modes :

1. **Organic** mode of development – talks about the projects that involve small development teams whose team members are familiar with the project and work to achieve stable environments. This category includes the projects like the *payroll systems*.

2. **Semi-detached** mode of development – talks about the projects that involve mixture of experienced team members in the project. This category includes the projects like the interactive banking system.
3. **Embedded** mode of development – talks about the complex projects that are developed under tight constraints with innovations in it and have a high volatility of requirements. This category includes the projects like the *nuclear reactor control systems*.

Drawbacks

- It is difficult to accurately estimate the KDSI in early phases of the project when most effort estimates are still not decided yet.
- Easily thrown misclassification of the development mode.
- Its success largely depends on tuning the model to the needs of the organization and this is done based upon the historical data which is not always available.

Advantages

- COCOMO is transparent that means we can se it working.
- Allows the estimator to analyse the different factors that affect the project costs.

Syllabus Topic : COCOMO II Model

7.6.2 COCOMO II Model

- COCOMO II constitutes of sub-models so as to produce in-detail software estimates. The sub models are listed as follows :
  - **Application composition model :** It is applied when software is being developed from existing parts.
  - **Early design model :** It is applied when system requirements are gathered and concluded but design has not yet started.
  - **Reuse model :** It is applied when software is being developed using the reusable components so as to compute the effort of integrating these components.
  - **Post-architecture model :** It is applied when once the system architecture has been designed and when more system information is gathered.

Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.

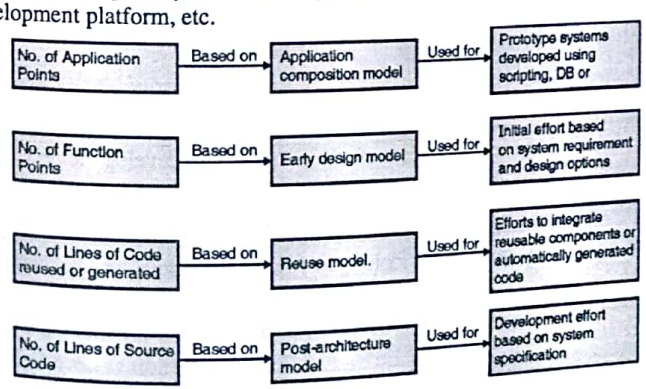


Fig. 7.6.1 : Use of COCOMO2 Model

- **Product attributes** describe the required characteristics of the software product being developed.
- **Computer attributes** describe the constraints imposed on the software product by the hardware platform.
- **Personnel attributes** describe the experiences and capabilities (of the project development team members) that are taken into account.
- **Project attributes** describe particular characteristics of the project.

– Estimating the calendar time required to complete a project and when staff will be required using a COCOMO2 formula

$$TDEV = 3 * (PM)^{(0.33+0.2*(B-1.01))}$$

– PM is the effort computation and B is the exponent (B is 1 for the early prototyping model). This computation predicts the nominal schedule for the project.

– The time required is independent of the number of people working on the project.

Improving Software Economics

– The software economics can be improved by using a 'balanced' approach as the key.

– The Five Key parameters that can help in improving the software economics are :

- Reducing the product size (Number of Lines of Code) *and* complexity of the software
- Improving the software development *process*
- Using more-skilled personnel i.e. improving the team effectiveness.
- Creating better environment by improving the *automation* with more appropriate tools and technologies.
- Achieving required quality by peer inspections

Table 7.6.1 : Trends on which the Five Key Parameters depend

Five Key parameters that effect the Cost Model	Trends
Size : describes abstraction and component based development technologies	High level programming Languages (C++, Java, VB, Object Oriented Methods and Visual modelling (analysis, design and programming) Reusability Commercial Exponents Packages
Process : involves methods and techniques	Iterative Development Process Maturity Models such as CMM Architecture-first development
Personnel : describes the effectiveness of the development team	Training to develop the personnel skills Team work Win-win culture
Environment : involves automated tools and technologies.	Integrated tools such as compiler, editors, and debuggers. Hardware performance Automated coding, documentation, testing and analyses.

Five Key parameters that effect the Cost Model	Trends
Quality : describes the Performance, reliability, accuracy issues	Hardware platform performance Peer inspections Statistical quality control

Syllabus Topic : Estimation for Agile Development

7.6.3 Estimation for Agile Development

In Agile development, the Project Management process is highly iterative and incremental, where all the project stakeholders actively work together to understand the domain, to identify what needs to be built, and to prioritize the functionality.

Estimation Scales in Agile development

- 1,2,3,5,8 (Fibonacci series)
- 1,2,4,8

Common techniques for Estimation in Agile development

- Expert opinion
- Analogy
- Disaggregation

Estimation Process

– User stories
- Users break down the functionality into "user stories"
- User stories are kept small and these include the acceptance criteria

– High level estimation
- After all the user stories are written, do a high level of estimation for setting the priorities.

– Story points
- Break down user stories to units of relative size so that you can compare their features which will be useful in measuring the size/complexity.

– Product backlog
- All story points which actually represent the project tasks are put into a bucket that acts as a Backlog which will have an item and its estimate
- This backlog is not time based, but point based

– Velocity
- Velocity is the number of story points completed per sprint
- Velocity is computed to predict how much work to commit to in a sprint and this only works if you estimate your story points' consistency

– Re-estimation
- As you complete more sprints, your velocity will go on changing. This change in Velocity is due to minor inconsistencies in the story point estimates
- Re-estimation of the entire project happens after each sprint.

Syllabus Topic : The Make/Buy Decision

7.6.4 The Make/Buy Decision

- Software engineering Managers face many problems in Make and Buy decisions that later have many number of acquisition options such as :
  - should software be purchased off the shelf ?
  - should it use "Full-experience" or "partial-experience" software components ?
  - should the software be custom built by an outside contractor ?
- The make/buy decision can be made based on the following conditions
  - Will the software product be available sooner if given to outside contractor than internally developed software?
  - Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
  - Will the cost of outside maintenance support be less than the cost of internal support?

Review Questions

Q. 1 Describe the cost estimation process.

Q. 2 Write short notes on cost estimation techniques.

Q. 3 List and describe the cost estimation parameters.

Q. 4 What is cost estimation and explain COCOMO model.

Q. 5 Explain the concept of estimation in agile development.

Q. 6 Explain the concept of Make/Buy decision

Q. 7 Explain in brief about : Software scope and feasibility

□□□

CHAPTER 8

UNIT II

Project Scheduling

Syllabus :

Basic Principles, Relationship between People and Effort, Effort Distribution, Time-Line Charts

8.1 Project Scheduling

Q. Explain the project scheduling process.

The *project schedule* is a calendar that is used to associate the tasks to be performed with the resources that will perform them. Before a project schedule is estimated, the project manager designs a work breakdown structure (WBS) which is an attempt to estimate the time needed to implement each task and the resources that are available for accomplishing each task.

Project Scheduling is dependent on project managers' intuition and experience.

Project Scheduling Process

- Divide the project into various tasks and estimate the duration and resources required to complete each task.
- Arrange the tasks so as to make optimal use of workforce.
- Reduce the task dependencies so as to avoid delays that might be caused by some task i.e. waiting for another to complete.

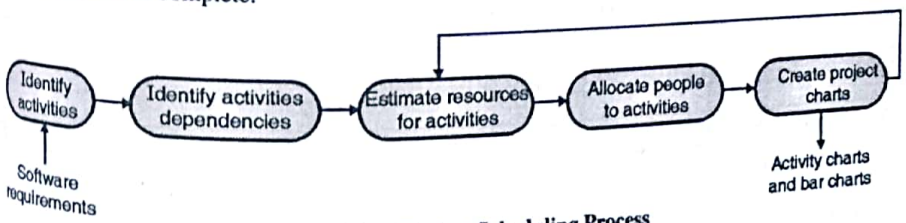


Fig. 8.1.1: Project Scheduling Process

Scheduling Problems

- Detecting the complexity of the problems and accordingly estimating the cost of developing a solution is difficult.

- Achieved productivity is not proportional to the number of people working on it.
- Adding new staff in the late project development moves the delivery date further far because of the communication gap between old staff and new staff.
- Always unforeseen events happen in the project development and these need to be considered in the planning.

Syllabus Topic : Basic Principles

8.2 Basic Principles

Q. List the basic principles of Project scheduling.

Project scheduling is about distributing or allocating the estimated efforts to specific software engineering tasks across the planed project duration.

Project scheduling builds a road map for the project manager when combined with estimation methods and risk analysis.

Basic principles guide software project scheduling:

1. Compartmentalization: The project must be categorized into number of manageable activities, actions and tasks by decomposing the process into sub processes.
2. Interdependency: Few compartmentalized activities, actions, or tasks occur in sequence where they are interdependent upon each other and while others occur in parallel.
3. Time allocation: Scheduling is very important where each task must be allocated some number of work units, defining their start date and completion date especially in case of the inter dependencies and also state whether work will be conducted on a full-time or part-time basis.
4. Effort allocation: The project manager must allocate number of people that must be are present at any given point of time.
5. Effort validation: The project manager must monitor that no more than the allocated number of people are present at any given point of time.
6. Defined responsibilities: Every task that is scheduled must be assigned to a specific team member.
7. Defined outcomes: Every task that is scheduled must have a defined outcome.
8. Defined milestones: Each set of tasks must be associated with a project milestone. And the milestone is said to be accomplished only when its associated tasks are reviewed for quality.

Syllabus Topic : Relationship between People and Effort

8.3 Relationship between People and Effort

Q. Explain the relationship between people and efforts in project scheduling

- A single person or just a small team of members is enough for working on small projects. But as the scope of the project increases, team size also increases. Adding unplanned and sometimes unskilled people in the later SDLC phases causes the schedules to slip even further. The people who are added later in the project should first of all understand the system properly which requires additional effort and time.
- Project schedules are flexible. As the project deadline gets closer and closer, you reach a point where the work cannot be completed on time, regardless of the number of people working on it.

And then a new delivery date is defined. If delivery is delayed, the PNR curve indicates that the project selling cost can be reduced substantially.

- Recommended *effort distribution* across the software process workflow is referred as 40-20-40 rule.
  - Front end analysis, design and back end testing are allocated 40% of all effort
  - 20% of effort is allocated to coding.
- Requirement analysis may comprise 40% of project effort.

Syllabus Topic : Effort Distribution

8.4 Effort Distribution

Recommended *effort distribution* across the software process workflow is referred as 40-20-40 rule.

- Front end analysis, design and back end testing are allocated 40% of all effort
- 20% of effort is allocated to coding.
- Requirement analysis may comprise 40% of project effort

8.4.1 Defining a Task Set

1. Consider the project type. Different types of projects

- Concept Projects: exploring new business concept or new application of technology
- New application development projects: new product requested by customer.
- Application enhancement projects: major modifications are done in functions, performance or interfaces.
- Application maintenance projects : correcting the existing functionality, adapting new functionality or extending existing software
- Reengineering projects : rebuilding all (or part) of a legacy system

2. Define the Degree of rigor. Various levels of degrees of rigor

- Casual : all framework activities are already applied, only minimum task set is required
- Structured: all framework and umbrella activities such as SQA, SCM, documentation, and measurement tasks are already streamlined.
- Strict: complete set of umbrella activities are applied to produce high quality products and robust documentation.
- Quick reaction: emergency situation and process framework is used but only tasks essential to good quality are applied.

3. Review rigor adaptation criteria. Various rigor adaptation criteria

- Project size
- Number of potential users
- Application durability
- Requirement stability
- Ease of communication with customer/developer

- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing
- Reengineering factors

4. **Determine task selector value.**
   - The task sector value is computed by adjusting the scores of different weight based project characteristics.
   - This computed task selector value is then used to select appropriate task set (casual, structured, strict) for the project.
5. **Consider concept development tasks.**
   - Determine the overall project scope
   - Preliminary concept planning such as establishing development team's ability to undertake the proposed work
   - Evaluating the technology risks in the software.
   - Proof of concept that depicts the feasibility of the technology in the software context
   - Concept implementation in a form i.e. used to sell to the customer
   - Customer's feedback on new technology i.e. implemented.

Syllabus Topic : Time-Line Charts

8.5 Time-line Charts

- Project scheduling involves preparing various graphical representations showing project activities, their durations and staffing.
- Graphical notations are used to illustrate the project schedule.
- Project scheduling is done by breaking down the whole development work into various tasks. Tasks must not be too small but should take about a week or two to accomplish them.

Table 8.5.1: Project Scheduling – Project Breakdown into Tasks

Activity	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1
T4	10	
T5	10	T2, T4
T6	5	T1, T2
T7	20	T1
T8	25	T4
T9	15	T3, T6
T10	15	T5, T7
T11	7	T9
T12	10	T11

- Following Activity chart describes the task dependencies and their critical path.

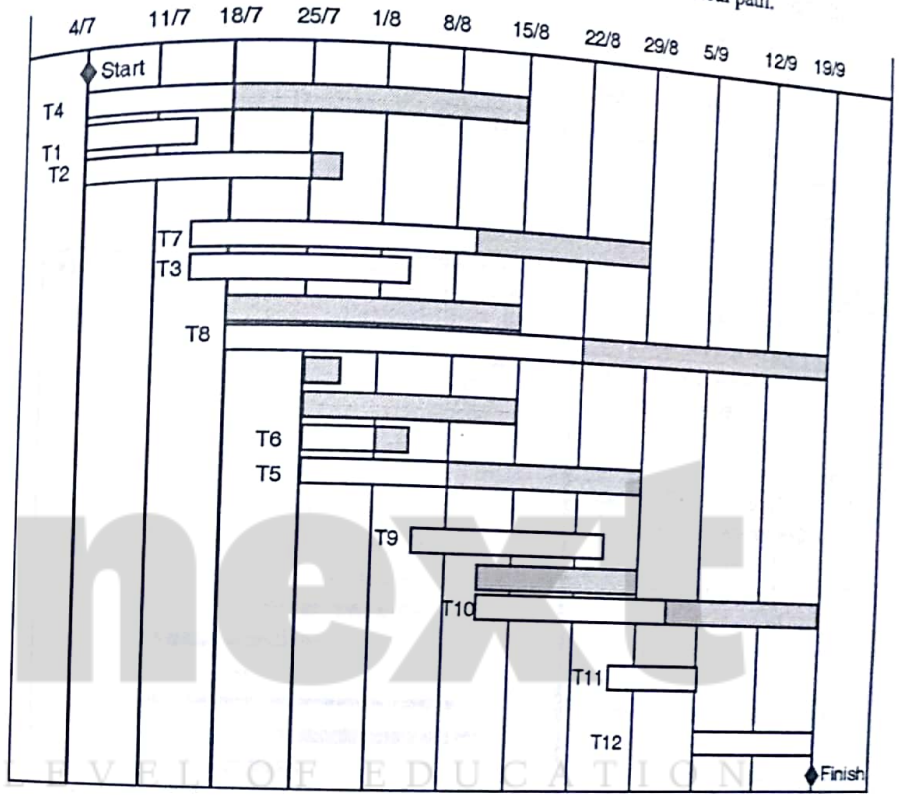


Fig. 8.5.1 : Project Scheduling - Activity Chart

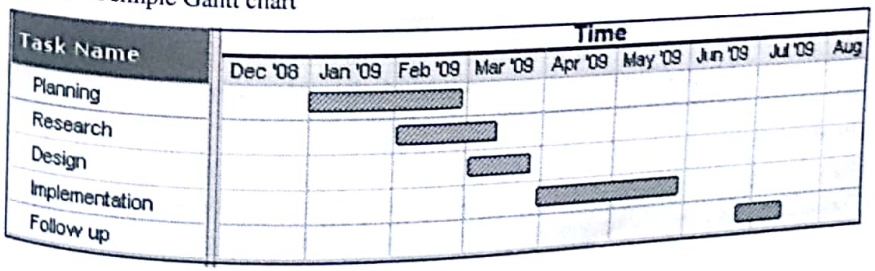
8.5.1 Gantt Chart

Q. What is the need of a Gantt chart and explain it with an example.

A Gantt chart used in project management is simply a graphical schedule that displays the activities (tasks or events) against time.

On the left of the Gantt chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the starting position of the bar reflects the start date, the length of the bar reflects the duration and the ending position of the bar reflects the end date of the activity.

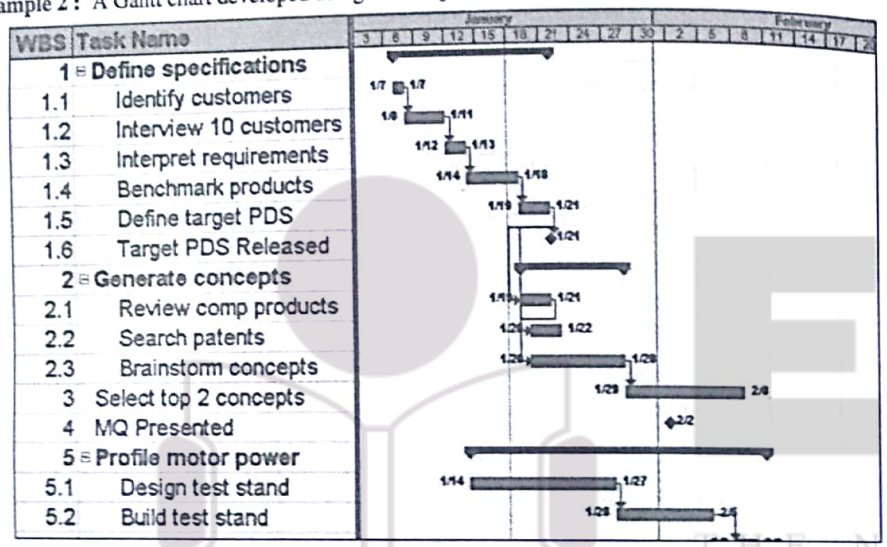
Example 1 : A simple Gantt chart



A Gantt chart is useful as it shows at a glance that

- What the various activities are ?
- When does each activity begin and end ?
- How long each activity is scheduled to last ?
- Where do the activities overlap with which activities do they overlap , and by how much time do they overlap ?
- The start and end date of the whole project

Example 2 : A Gantt chart developed using MS Project tool



Symbols used in the above Gantt Chart

- **Milestone :** These are represented by black diamonds which represent a significant event on a project with zero duration.
- **Summary Tasks :** These are represented by thick horizontal black bars with arrows at beginning and end. The WBS activities are referred to as tasks.
- **Individual tasks :** The light gray horizontal bars represent the duration of each individual task.
- **Relationships or Dependencies :** Vertical arrows connecting these symbols show relationships or dependencies between tasks.

Benefits

- The Gantt chart shows deadlines and some of the sequential nature of the tasks.
- It evaluates progress on a project by showing actual schedule information.
- It gives a comparison of planned and actual project schedule information. The planned schedule dates for activities are called the baseline dates.

Drawbacks

- The Gantt chart is incapable in identifying mid process deadlines and bottlenecks in the process.
- They often do not show relationships or dependencies between tasks.

8.5.2 PERT/CPM Chart

Q. Explain a PERT/CPM chart with an example.

- PERT (Programme Evaluation and Review Technique) and CPM (Critical Path Method) are the scheduling techniques used to represent the network of activities carried out in a project.
- PERT and CPM both are used to determine the critical path and are based on the network representation of activities and their scheduling. This determines the most critical activities to be performed so as to meet the completion date of the project.

A PERT/CPM chart is similar to a flow chart having lines (arrows) representing activities/tasks and nodes representing events or milestones.

These arrows go from left to right. Nodes represent the end and beginning of sequential activities.

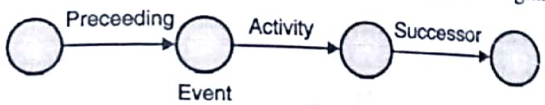


Fig. 8.5.2

Arrows departing from a node indicate parallel activities.

Example :

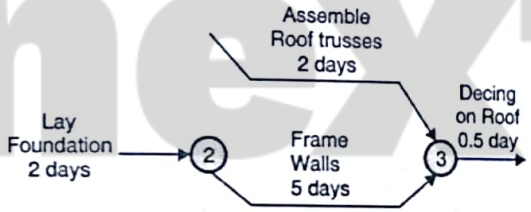


Fig. 8.5.3

This chart shows ;

- Assembly of roof trusses and framing of the walls can be done at the same time in parallel.
- Decking on the roof requires the both assembly of the roof trusses and framing of walls
- Assembling the roof trusses is not dependent upon the Foundation but Framing the walls is.

Both PERT and CPM provide quantitative tools that allow the software planner

1. **Determine the 'critical path' :** It is a chain of tasks that reflect the duration of the project.
2. **Estimate 'most likely time'** required to accomplish the individual tasks by applying statistical models.
3. **Calculate 'boundary times'** that define a 'time window' for a particular task where the ***Slippage*** in the design of one function can retard further development of other functions and the ***Riggs*** describe important boundary times that may be discerned from a PERT or CPM network:
   i. The earliest time that a task can begin when all preceding tasks are completed in the shortest possible time.
   ii. The latest time for task initiation before minimum project completion time is delayed.
   iii. The earliest finish is the sum of the earliest start and the task duration.

iv. The latest finish is the latest start time added to task duration.

v. The total *float* is the amount of surplus time allowed in scheduling the tasks so that the network critical path is maintained on schedule.

Calculating the Boundary time helps in determining the critical path and providing quantitative method for evaluating the processes as tasks complete.

8.5.2(A) Difference between PERT and CPM

Q. Give the difference between PERT and CPM

Table 8.5.2: Difference between PERT and CPM

Sr. No.	PERT	CPM
1.	Program Evaluation and Review Technique	Critical Path Method
2.	Developed by the US Navy with Booz Hamilton Lockheed on the Polaris Missile / Submarine program 1958.	Developed by El Dupont for Chemical Plant Shutdown Project- about same time as PERT
3.	Three estimates are used to form a weighted average of the expected completion time of each activity based on probability distribution of completion time.	Only one estimate of completion time of each activity is used.
4.	It is a tool used for planning and control of time.	It is a tool used for cost estimation apart from estimating time.
5.	It is used for one-time projects involving activities of non-repetitive nature (i.e. activities that are never performed before) in which time estimates are uncertain such as installing a new information system. It is used where times cannot be estimated with confidence.	It is used for completion of projects involving activities of repetitive nature. It is used where times can be estimated with confidence, familiar activities.
6.	Identifies critical path and activities, guides in monitoring and controlling the project.	Same as PERT
7.	Meeting the time target or estimating the percent completion is more important.	Minimizing cost is more important.
8.	Example: Involving new activities or products, research and development etc.	Example: construction projects, building one off machines, ships, etc.

8.5.2(B) Use of PERT & CPM Chart

Benefits of PERT/CPM chart are :

- It is useful in planning large projects. The sequential tasks can be analysed to determine the critical path and predict total project length.
- These techniques help in translating the high complex projects into a set of simple and logically arranged activities thereby;
  - Clarifying the thoughts and actions

  - Developing clear and unambiguous communication from top to bottom and vice versa among the people responsible for executing the project.
- These techniques provide a detailed analysis of network of activities that help project manager to early detect the difficulties that are expected to crop up during the course of execution in future thereby;
  - Minimizing the delays and holdups that may occur during execution in future
  - Take corrective actions well in time
- These techniques isolate the activities which control the project completion and therefore result in expeditious completion of the project.
- These techniques divide the responsibilities and improve the coordination between the different departments.
- These techniques help in timely allocation of resources to various activities to achieve optimal utilization of resources.

8.5.2(C) Solved PERT/CPM Network Diagrams

Example 8.5.1 :

Consider the list of four activities for making a simple product :

Activity	Description	Immediate predecessors
A	Buy Plastic Body	–
B	Design Component	–
C	Make Component	B
D	Assemble products	A,C

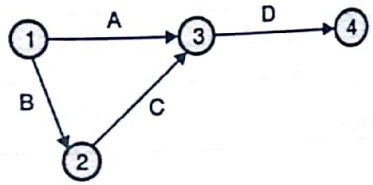
(Immediate predecessors for a particular activity are the activities that, when completed, enable the start of the activity in question.)

Solution :

Sequence of Activities

- Activities A and B can start at anytime, since neither of these activities depend upon the completion of prior activities.
- Activity C cannot be started until activity B has been completed
- Activity D cannot be started until both activities A and C have been completed.

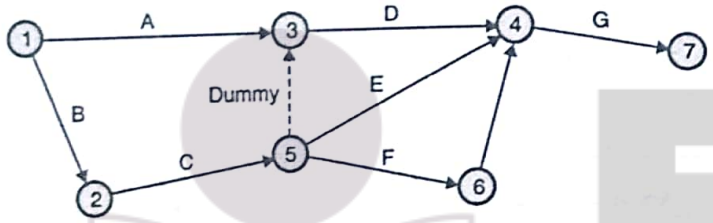
Network Diagram



Example 8.5.2 : Develop the network for a project with following activities and immediate predecessors.

Activity	Immediate predecessors
A	–
B	–
C	B
D	A,C
E	C
F	C
G	D,E,F

Solution :



D, E, and F as the immediate predecessors for activity G.

Dummy activities is used to identify precedence relationships correctly and to eliminate possible confusion of two or more activities having the same starting and ending nodes.

Dummy activities have no resources (time, labor, machinery, etc) purpose is to PRESERVE LOGIC of the network.

Example 8.5.3:

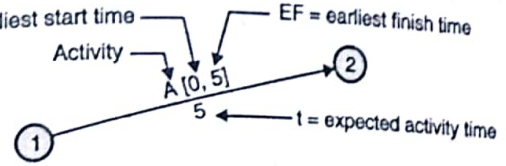
Draw the network diagram for this information that indicates that the total time required to complete activities is 51 weeks.

Activity	Immediate predecessors	Completion Time (Week)
A	–	5
B	–	6
C	A	4
D	A	3
E	A	1
F	E	4
G	D, F	14
H	B, C	12
I	G,H	2
	Total	**51**

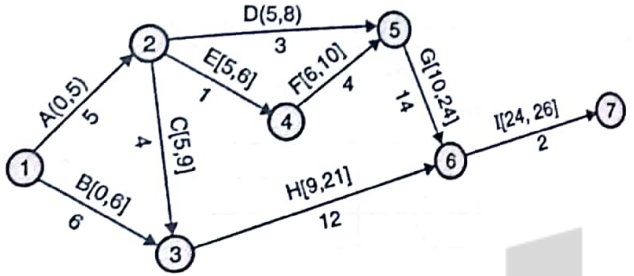
Solution :

- The earliest finish time is calculated as EF = ES + t.
  - For example, for activity A, ES = 0 and t = 5; thus EF = 0 + 5 = 5

A sample Arc with ES and EF time :

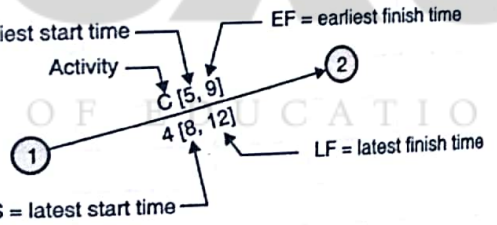


Network diagram with ES and EF time

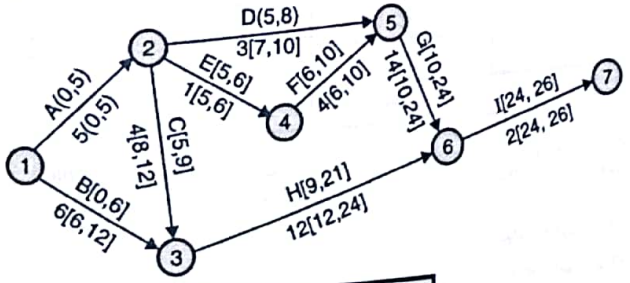


- The latest start time for each activity is calculated as; LS = LF – t
- For example, for activity I, LF = 26 and t= 2, thus the latest start time for activity I is LS = 26 – 2 = 24

A sample Arc with ES and EF time, LS and LF time



Final Network diagram with ES and EF time, LS and LF time



Review Questions

Q.1 Explain the project scheduling process.

Q.2 List the basic principles of Project scheduling.

Q. 3 Explain the relationship between people and efforts in project scheduling

Q. 4 Describe the effort distribution in project scheduling in brief

Q. 5 What is the need of a Gantt chart and explain it with an example.

Q. 6 What is the need of PERT/CPM chart.

Q. 7 Give the difference between PERT and CPM

Q. 8 Explain a PERT/CPM chart with an example

Q. 9 For the following data, draw the PERT/CPM network diagram.

Activity	Preceding Activity
A	-
B	A
C	A
D	B,C
E	B
F	D, E

Q. 10 Draw the network diagram for the following problem.

Activity	Preceding Activity	Time(days)
A	-	10
B	-	14
C	A	8
D	A	7
E	B	5
F	B	10
G	C	9
H	D, E	11
I	G, H	5

Q. 11 For a project that includes tasks like requirement analysis, designing, coding, testing and implementation; break the work down into small enough parts that you can estimate the time for completion.

Assignment 1: Build a WBS chart

Assignment 2: Use the WBS chart to create a Gantt chart.

Assignment 3: Use WBS chart to create a PERT/CPM Chart.

□□□

UNIT III

CHAPTER 9

Risk Management

Syllabus

Software Risks, Risk Identification, Risk Projection and Risk Refinement, RMMM Plan.

Syllabus Topic : Software Risks

9.1 Software Risks

- A risk is a probability of an occurrence of some adverse circumstance.
- Risks are categorized into three paths:
  1. Important Risks but not resolved
  4. Unimportant Risks that are not resolved and later ultimately change into Important Risks.
  3. Unidentified Risks that later become Important Risks
- In order to manage feasibility throughout the project, risk management is needed.
- Risk management seeks to balance risk and reward.
- Different organizations are more or less adverse to risk, i.e. They are willing to take greater risks in order to achieve greater rewards.
- Software Risks possess two characteristics :
  1. **Uncertainty :** The risk may or may not happen; i.e. there are no 100% probable risks.
  2. **Loss :** The risk becomes a reality, unwanted consequences or losses may occur.
- It is important to measure the level of uncertainty and degree of loss associated with each risk.

The two basic kinds of Risks

1. **Predictable risks :** These are caused due to staff turnover, poor communication with customer, dilution of staff performance.
2. **Unpredictable risks :** These are extremely difficult to identify in advance.

9.1.1 Types of Risks

Q. Explain the different categories of Risks.

- **Project risks** affect project schedule and resources. These effect the project plan and resources. If these turns into a reality, it is likely that project schedule will fall and that costs will rise. They may affect the budget, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.
- **Product (Technical) risks** affect product's quality and performance. These effect the quality and timeliness of the software to be produced. Implementation becomes difficult if this risk turns into a reality. They may affect design, implementation, interface, verification and maintenance problems. Also some of the risk factors include ambiguity in specifications, technical uncertainty, technical obsolescence (not used anymore).
- **Business risks** affect development organisation and procuring organization. These affect the viability of the software to be built. Business risks often create confusion in the project or the product. The top 5 business risks are:
  - **Market risk :** Building an excellent product and selling it in the market where no one is in need of it
  - **Strategic risk :** Building a product that no longer fits into the overall business strategy.
  - **Sales risk :** Building a product where the sales team does not know how to sell it.
  - **Management risk :** building a product with the least support of senior management.
  - **Budget risk :** building the product which is out of budget or personnel commitment.

Table 9.1.1 : Software Risks

Reason of Risks	Risk Type	Description of Risks
Change in Staff	Project risk	Experienced and skilled staff leaves the project or organization before it is finished.
Change in Management	Project risk	Changes in various priorities lead to various changes in the Organisation.
Hardware unavailability	Project risk	Hardware on which the project has to be developed or deployed for test is not delivered on time.
Change in Requirements	Project and Product risk	Large number of changes in the requirements in later phases.
Specification delays	Project and Product risk	Specifications of necessary interfaces are not available on time.
Underestimating the size	Project and Product risk	The size of the system is underestimated which effects the scope, cost and schedule.
Poor performance of CASE tools	Product risk	CASE tools necessary for project development don't perform as expected.
Change in Technology	Business risk	The technology on which the system is built is outdated by some new technology.
Competition of Product	Business risk	Another competitive product is marketed before our product delivery and deployment is completed.

9.2 Seven Principles of Risk Management

Q. Explain the seven principles of Risk Management.

1. **Maintain a global perspective**

   View the software risks within the system context and business problem.

2. **Take a forward-looking view**

   Think about the risks that may arise in the future and establish precautionary plans so that those risks even on existing can be manageable.

3. **Encourage open-communication**

   Encourage all stakeholders and users to suggest risks at any time. Consider them even if told in an informal manner.

4. **Integrate**

   Identify and consider those risks, which have to be integrated into the software process.

5. **Emphasize a continuous process**

   Throughout the software process, the risk management team should be vigilant enough to modify identified risks as and more information is known and add new ones if more information is added.

6. **Develop a shared product vision**

   Sharing the same vision of the software by all stakeholders can do better risk identification and assessment.

7. **Encourage teamwork**

   The talents, skills and knowledge of all stakeholders should be pooled to get better idea about the risks involved.

9.3 Risk Management Process

Q. Write short notes on : Risk Management.
Q. What is Risk Management? Explain different stages involved in Risk Management.

- It is about identifying risks to ensure that these risks do not turn into major threats.
- It is about identifying risks and deriving plans to reduce their effect on the project.

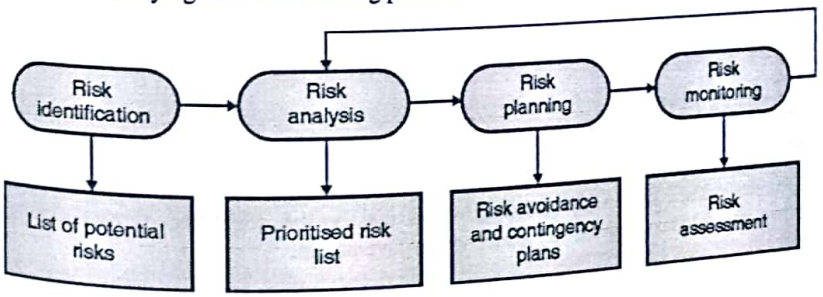


Fig. 9.3.1 : Risk Management Process

Syllabus Topic : Risk Identification

9.3.1 Risk Identification

Q. Explain Risk Identification.

– It is about identifying the project, product and business risks. Risk identification can start with source of problem or with the problem itself.

Risk	Affects
Technology	– Database used in the system may be poor which cannot process as many transactions per second as expected. – Reusable software components might contain defects that limit the system's functionality.
Staff	– It is difficult to recruit skilled staff. – Key staff are unavailable at critical times may because they are ill or out of station. – Required training is not given to the staff.
Organizational	– One project is handled by number of managerial groups of the organization. – Organizational financial problems force cut-off in the project budget.
Case Tools	– Insufficient code generated by CASE tools. – Not able to integrate the designs in CASE tools.
Requirements	– Changes in requirements lead to major changes in designs that lead to rework – Customers fail to understand the side effect of frequent changes in requirements on the project.
Estimation	– Time required to develop the software is underestimated. – Size and scope of the software is underestimated. – Defect Repair rate is underestimated.

Risk Identification can be done by two ways of analysis

(a) **Source analysis**

The target of risk management is to identify the sources of risk either internal or external to the system.

Examples : project stakeholders, company's employees.

(b) **Problem Analysis**

The threats which exist with various entitled mainly the Identified threats dealing with shareholders, customers, legislative bodies such as government.

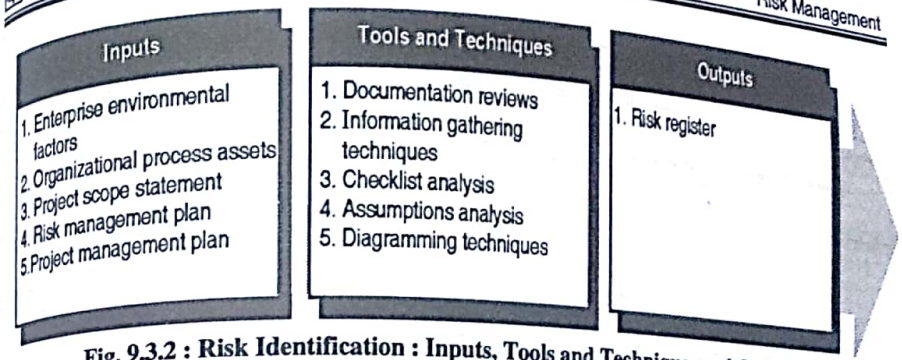


Fig. 9.3.2 : Risk Identification : Inputs, Tools and Techniques and Outputs

Risk Identification Methods

Q. Explain the methods involved in it

The Risk Identification methods depend on industry, practice, culture and compliance.

1. Objectives-based

Any event that may endanger an organization or a project team in achieving their objective partly or completely.

2. Scenario-based

Scenarios are designed so that they can be an alternative way to achieve an objective.

3. Taxonomy-based

Based on taxonomy (breakdown of possible risk sources) and knowledge of best practices, a questionnaire is compiled. The answers to questions reveal risks.

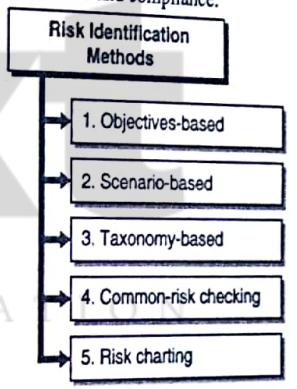


Fig. C9.1 : Risk Identification Methods

4. Common-risk checking

For application in a particular situation, each risk in the common list maintained by several industries can be checked.

5. Risk charting

This method combines the above approaches y listing resources at risk, threats to them, modifying facts which may increase/decrease the risk and consequences which can be avoided.

9.3.2 Risk Analysis

- Assess the probability and consequences of these risks.
- Probability scale ranges from *very low, low, moderate, high or very high.*
- Risk effects may range from *catastrophic, serious, and tolerable to insignificant.*

Table 9.3.2 : Risks, their Probability and Effects

Risk	Probability	Effects
Organizational financial problems force cut-off in the project budget.	Low	Catastrophic
It is difficult to recruit skilled staff.	High	Catastrophic
Key staff are unavailable at critical times	Moderate	Serious
Reusable software components might contain defects that limit the system's functionality.	Moderate	Serious
Changes in requirements lead to major changes in designs that lead to rework	Moderate	Serious
One project is handled by number of managerial groups of the organization	High	Serious

Risk analysis helps to define preventive measures to reduce the probability of risks and identify counter measures to successfully deal with these risks.

9.3.3 Risk Planning

Q. Write short notes on : Risk Planning.

- A risk management plan is a document prepared by a project manager to estimate the risks, to estimate the consequences and seriousness of these risks and to create response plans to avoid/solve them.
- A risk management plan contains an analysis of likely risk with both high and low impact as well as mitigation strategies to help the project avoid being derailed when any common problems arise.
- A risk management plan should be periodically reviewed by the project team so that the analysis does not become stale and not reflective of actual potential risk.
- It includes the following tasks :
  - Draw up plans to avoid or reduce the risks.
  - Design strategies to handle the risks.
  - Design avoidance strategies to avoid the probability of risk occurrence.
  - Design minimization strategies to reduce the probability of risk occurrence.
- Design Contingency plan so that if the risk arise, contingency plan is used to deal with that risk.

Table 9.3.3 : Risks and Strategies to overcome the risks

Risk	Strategy to overcome the risk
Organisational financial problems	Prepare a briefing document for senior management showing how the project makes an important contribution in achieving the goals of the business.
Requirement Change problems	Warn the customer regarding the potential difficulties caused by specification delays.
Staff Unavailability problems	Reorganise the development team so that there is more overlap of work and therefore, team members will more easily understand each other's jobs.
Defective Reusable components	Replace potentially defective components with components of known reliability.

- The 4 potential strategies, which are included in the Risk Management Plan, are :
  1. **Accept risk :** assume that the negative impact occurs.
  2. **Avoid risk :** Change plan to prevent the problem
  3. **Mitigate risk :** minimizing its impact through intermediate steps.
  4. **Transfer risk :** Outsource risk management taks to a third party.

9.3.4 Risk Monitoring and Control

Q. Write short notes on : Risk Monitoring.

- It is about monitoring and controlling the risks throughout the project development.
- Assess each and every identified risk regularly so as to check whether or not it is becoming less or higher.
- Assess whether or not the effects of the risk have reduced.
- Discuss each key risk in the management progress meetings.
- To overcome the risks, *line of Business Organization* is maintained.
- It is the process of identifying, analyzing and planning to overcome the newly discovered risks and managing identified risks.

Inputs to Risk Monitoring and Control

- **Risk management plan :** Plan to approach and manage project risks.
- **Risk register :** Comprehensive risk listing for projects.
- **Approved changer requests :** Necessary adjustments to work methods, contracts, project scope, and project schedule.
- **Work performance information :** Status of scheduled activities being performed to accomplish project work.
- **Performance reports :** Project's performance with respect to cost, scope, schedule, resources, quality and risk are shown using bar charts, s-curves, tables, histograms to organize and summarized information such as earned value analysis and project work progress.

Best practices followed in Risk monitoring and control

1. Risk reassessment

Risks are reassessed on a regular scheduled basis. Reassessment enables the project manager to evaluate risk impact, risk rating (urgent), determine old or new risks. These are normally addressed at status meetings.

2. Status meeting

Team members share their experiences in this forum and inform other members about their progress and plans and this is mostly done through open discussions in the presence of project manager.

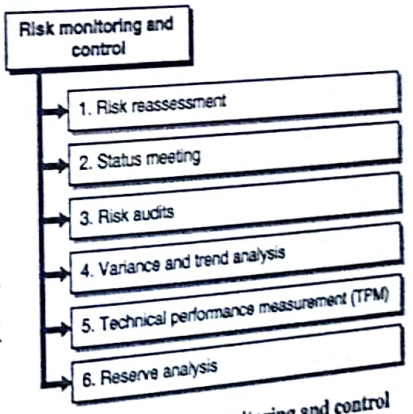


Fig. C9.2 : Risk monitoring and control

➔ **3. Risk audits**

They examine and document the effectiveness of planned risk responses and their impact on schedule and budget. Performed by risk auditors who specialize in risk assessment and auditing techniques. They are from outside the project team.

➔ **4. Variance and trend analysis**

Variance analysis = planned budget - actual budget / schedule.

Trend Analysis = project performance / time. This tells if performance is getting better or worse.

➔ **5. Technical performance measurement (TPM)**

Identifies deficiencies in fulfilling the system requirements, provides early warning of technical problems and monitor the technical risks.

➔ **6. Reserve analysis**

Compares contingency measures and the remaining amount of risk to ascertain if there is enough reserve in the pool. Contingency measures are buffers of time, funds or resources set aside to handle risks that arise as project grows.

Syllabus Topic : Risk Projection

9.4 Risk Projection (Estimation)

Q. Write short notes on : Risk Projection.

Risk projection is also known as risk estimation. It rates the risks based on two ways :

1. Probability that the risk is real
2. Consequences of the problems associated with that risk

☞ **Steps in Risk Projection (Estimation)**

The project planner performs four risk projection activities :

1. Define a scale that reflects the probability of the risks. For example, 1-low … 10-high.
2. Define the consequences of the risks
3. Project (estimate) the impact of the risk on the project and the product
4. Define the overall accuracy of the risk projection

9.4.1 Risk Table

Risk table consists of five columns which is helpful in efficient risk projection.

1. **Risk Summary :** short description of the risk. List all risks in the first column
2. **Risk Category :** Mark the category of each risk
3. **Probability :** Estimate the probability of each risk occurring based on group input
4. **Impact :** Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value : (1) catastrophic (2) critical (3) marginal (4) negligible. Sort the rows by probability and impact in descending order
5. **RMMM :** Risk Mitigation, Monitoring, and Management Plan

☞ **Impact of Risks**

Three factors of risks that affect the consequences are :

1. **nature :** indicates the problems that are likely to happen if the risk occurs
2. **scope :** combines the severity of the risk with its overall distribution
3. **timing :** considers when and for how long the impact will be felt

Overall **risk exposure** formula is RE = P x C

where,

- P : probability of occurrence for a risk
- C : cost to the project if the risk actually occurs

Example :

Consider the Probability : 80%

Cost estimation : If 90 reusable software components were planned. If only 60% of it can be used, 36 components need to be developed from scratch. Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $12.00, therefore the overall Cost (impact) to develop the components would be 36 x 100 x 12 = $43,200

Risk exposure : RE = P X C = 0.8 x 43,200 ~ $34,560.

Syllabus Topic : Risk Refinement

9.5 Risk Refinement

Q. Write short notes on : Risk Refinement.

- The process of restating the risks in more details which will be easier to mitigate, monitor, and manage is called Risk Refinement.
- In this process, the risks are represented and detailed in the CTC (condition-transition-consequence) format. This shows that; for the given <condition> then there is a possible <consequence>.

☞ **Example of detailing the risk using CTC format**

Given ***condition*** that all reusable software components must conform to specific design standards and that some do not conform, then there is ***concern (possibly consequence)*** that only 60% of the planned reusable modules may actually be integrated into the built system, resulting in the need to develop the remaining 40% of components.

Syllabus Topic : RMMM Plan

9.6 RMMM Plan

Q. Explain RMMM plan in detail.

RMMM stands for Risk Mitigation, Monitoring, and Management. RMMM is an effective strategy for dealing with risks.

☞ **Mitigation**

It is a primary strategy and is achieved through a proper planning.

For example : risk of high staff turnover.

Strategy for reducing staff turnover

- Discussions with current staff to identify the causes for reduced turnover for example, poor working condition, low pay, competitive market.
- Mitigate those causes that are under our control before the project begins.
- Organize project teams such a way that you can monitor the development activities of each team.
- Document the standards and establish mechanisms so as to ensure that documents are developed in a timely manner.
- Conduct peer review of all the work.
- Plan a backup for every staff member and for every complex technology and tool.

Monitoring

The project manager monitors the risks :

- Access predicted risk
- Assure about risk aversion
- Collect useful information for future risk

Risk monitoring has three objectives

1. To assess whether predicted risks occur
2. To ensure that risk avoidance steps are taken
3. To collect information i.e. useful in future risk analysis

The findings from risk monitoring helps the project manager to know which risks caused which problems throughout the project and whether the risks have decreased or still the same.

Management

- Risk management and contingency planning assume that mitigation efforts have failed and that the risk has become a reality.
- Prepare RMMM plan. The RMMM plan can be the part of an SDLC plan or can be a separate document.
- RMMM plan is a document of all work performed as a part of risk analysis and used by project manager as a part of overall project plan
- Once RMMM has been documented and the project has begun, the risk mitigation, and monitoring steps begin
  - Risk mitigation is a problem avoidance activity
  - Risk monitoring is a project tracking activity

Steps of risk management

- Risk assessment - Identify, Analyze and priories
- Risk control - Planning, Resolution and monitoring.

MCQs

Q. 1 Which of the following would you use to help identify risks? (Select as many as apply)

(a) Contractual requirements or statements of work (SOWs)

(b) Supplier contracts or customer agreements (c) Project plan assumptions

(d) Lessons learned files from previous projects

(e) Project schedule

Q. 2 Risk exposure is a combination of what two things?

(a) Probability and impact (b) Impact and severity

(c) Frequency and impact (d) Severity and frequency

Q. 3 What is included in a list of identified risks?

(a) Probability (b) Impact

(c) Frequency (d) Both probability and impact

Q. 4 The primary objective of risk response planning is to:

(a) Determine what, if anything should be done with a risk.

(b) Take the guesswork out of the risk response management process.

(c) Compare the cost of risk response to the expected monetary value.

(d) Improve the accuracy of risk assessment.

Q. 5 Which of the following are factors that affect the selection of a risk mitigation strategy?

(a) Customer satisfaction

(b) The project manager's authority, accountability, and ability

(c) Commitment from the project manager and upper management

(d) All of the above

Q. 6 How is the project risk exposure determined?

(a) By turning it into a known risk

(b) By determining whether it is a business risk or a pure risk

(c) By analyzing its probability and impact

(d) By determining what part of the life cycle it is in

Q. 7 Match the type of risk with its definition.

(a) Business - A. Normal risk of doing business with opportunity for gain or loss. For example, the release of a new product.

(b) Pure - B. Risk that can be anticipated. These risks can be managed and controlled. For example, sales slowdown after Christmas.

(c) Known - C. Risk that presents an opportunity for loss only. For example, an earthquake.

(d) Unknown - D. Risk not planned for and not recognized. For example, release of a competitive product.

Q. 8 _____ is a framework for the iterative process of planning, tracking, and reacting to risk.

(a) Risk analysis (b) Risk management

(c) Project management (d) Project analysis

Q. 9 Which step in the risk management process defines the probability and impact of each risk to determine the severity ?

(a) Risk Identification (b) Risk Analysis

(c) Risk Planning (d) Risk Tracking and control

Q. 10 Which of the following is true about risk identification?

(a) Risk events should be identified early and projects reviewed regularly to ensure that no significant risk events are missed.

(b) The risk identification process should be repeated on a regular basis as part of your key checkpoints.

(c) Be extremely detailed about possible risks, even if they do not at first seem realistic risks; this allows for coverage of all possibilities.

(d) Start a new risk register at the beginning of each project phase.

Q. 11 Which of the following is done to create a risk management plan?

(a) Accept that risk might not change any of the project elements.

(b) Identify risks, analyze risks, and define what, if anything, should be done about the risks.

(c) Assign the actions to project team members.

(d) Both B and C.

Q. 12 Which of the following is true about the importance of risk management ?

(a) It ensures that the project will be on budget.

(b) It helps reduce the impact of risk in a project.

(c) It eliminates the impact of all risk.

(d) It ensures that the project will be on schedule.

Review Questions

Q. 1 Write short notes on :

(a) Risk Management (b) Risk Planning

(c) Risk Monitoring (d) Risk Projection (e) Risk Refinement

Q. 2 Explain the seven principles of Risk Management.

Q. 3 Explain Risk Identification and the methods involved in it

Q. 4 Explain the different categories of Risks.

Q. 5 Explain the different types of risks?

Q. 6 What is Risk Management? Explain different stages involved in Risk Management.

Q. 7 Explain RMMM plan in detail.

□□□

UNIT III

CHAPTER 10

Software Quality Assurance

Syllabus

Elements of SQA, SQA Tasks, Goals, Metrics, Formal Approaches to SQA, Six Sigma, Software Reliability, The ISO 9000 Quality Standards, Capability Maturity Model.

10.1 Quality

Definition

"Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected from all professionally developed software."

10.1.1 Quality Perceptions

The software is developed by an engineer, it is required by the customer, and is used by various end users. Thus, each of them has a different perception about quality that decides whether the software is good or not.

- Engineer may judge :
  - User satisfaction,
  - Portability,
  - Maintainability,
  - Robustness
  - Efficiency
- Customer may judge : cost
- User may judge :
  - Reliability (i.e. No Defects that may either stop computing or may produce unexpected results),
  - Usability.

10.1.2 Importance of Software Quality

Q. Why is software quality important ?

- Aim of Quality Measure is : **Satisfying Customer**

- Why Quality is important can be inferred from the following reasons :
  - **Quality is a competitive issue :** Our product's quality should be better than our competitor's product.
  - **Quality is must for Survival :** Just taking advantage of your so called Fame and good marketing is not enough to survive in the market; Quality of your product decides your survival in the market.
  - **Quality gives entry into international Market :** Only quality products are given ISO marks and permission to enter in the global market.
  - **Quality is cost effective :** If the quality is maintained from the initial phases of SDLC, then the cost expenditure in correcting the risks or errors is very less as compared to the maintenance cost of the defects detected after the delivery of the product.
  - Quality helps retain customers & increase profits.
  - Quality is the hall-mark of world class business.

10.1.3 Quality Challenges

1. Quality of software should be given importance from the very initial Stage of Software Development, not just at the end of coding.

   Example : If a pharmacist is making a new medicine and if its sample is not tested in the chemical laboratory from the beginning of its development and if it is delivered to the stores *without verifying and validating it's quality*, then it may lead to the death of lots of people + the financial and reputational loss to the company.
2. Monitor not only individual segments (complete or complex or large system is divided into simple or individual systems) but also interaction between the segments.

   Example : Consider a subject (complete system) with different chapters (individual segments) – Each chapter individually has a meaning. But check out whether each chapter is related (interaction between segments) to the other or not. Only if individual chapters have a relative meaning with each other, only then they collectively make a subject. In an English Literature subject, you cannot insert a maths chapter.
3. The Quality criterion differs from phase to phase of Software Development.

   Example : Precise Req., Detailed SRS, Cost and Time Estimation, Risk Analysis are the quality criteria's at Requirement Analysis phase; Coupling & Cohesion in Design phase; Code Efficiency & Reusability in Coding phase.
4. Quality measures used for small systems may not be appropriate for larger ones.

   Example : Indica car is a very good and quality car but can be used to accommodate 5 people not to accommodate 10 people.

10.1.4 Causes of Errors In a Software Product

Q. What are the various causes behind the errors in a software product ?

- Incomplete or erroneous specification
- Misinterpretation of customer communication
- Intentional deviation from specification
- Violation of programming standards
- Error in data representation

- Inconsistent module interface
- Error in design logic
- Incomplete or erroneous testing
- Inaccurate or incomplete documentation
- Error in programming language translation of design
- Ambiguous or inconsistent human-computer interface
- Miscellaneous.

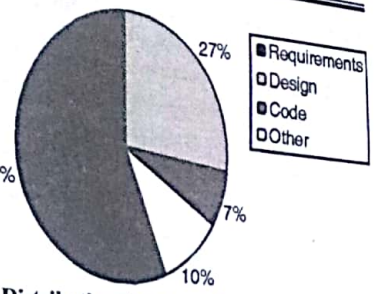


Fig. 10.1.1 : Distribution of Defects/Errors in the SDLC

The pie chart (Fig. 10.1.1) defines the defects that are caused at different phases of Software Development Life Cycle (SDLC) :

10.2 Quality Control and Quality Assurance

Q. State the difference between quality control and quality assurance.

- Quality Assurance is the process of defining how software quality can be achieved and how the development organization knows that the software has the required level of quality.
- *Quality control* and *Quality assurance* are the essential activities for any organization in the process of developing a product.

Table 10.2.1 : Software Quality Control v/s Software Quality Assurance

Sr. No.	Software Quality Control	Software Quality Assurance
1.	It is a *corrective approach* – corrects the faults when they occur.	It is an *preventive approach* – prevents the faults from occurring by providing rules and methods.
2.	*Finds defects* and then corrects it.	*Prevents defects* from occurring.
3.	It is a task conducted on the *product.*	It is a task conducted in the *process.*
4.	Gives confidence to *producer.*	Gives confidence to *customer.*
5.	Process of comparing the product quality with applicable standards and taking appropriate action when non-conformance is detected.	It is a set of planned and systematic set of activities that provides adequate confidence about establishing the requirements properly and assures that the products or services conform to specified requirements.
6.	Only tester is responsible for QC.	Entire team is responsible for QA.
7.	Detects and reports and corrects the defects.	Prevents the introduction of issues or defects.
8.	Examples: Walkthrough, inspections, testing conducted on the documents or on software product.	Examples : Defining process, automated engineering and testing tools i.e. CASE (engineering) and CAST (testing) tools, training, quality audits are used in development process.

10.2.1 Changing View of Quality

- The first quality assurance and control function was introduced in Bell labs in 1916 and spread rapidly throughout the manufacturing company across the world.
- Before twentieth century, quality control was not given that importance. It was considered as the sole responsibility of developers. The Table 10.2.2 describes the changing view of quality:

Table 10.2.2 : Changing view of Quality

Sr. No.	PAST	PRESENT
1.	Quality was considered as the responsibility of developers	Quality is considered as the responsibility of everyone's responsibility
2.	Defects are intentionally hidden from customers and management so hat developers are not burdened with extra work of correcting these errors.	Defects are highlighted and brought to surface so as to conduct corrective measures and achieve user satisfaction
3.	Quality problems led to blame each other and giving faulty justifications and excuses. This only led to quarrels among the team members and an unhealthy team work.	Quality problems lead to co-operative solutions so as to prevent the failures.
4.	Not much work was welcomed. Say, for example, correctness measures accompanied with minimum documentation.	Documentation is considered as essential to note down the lessons learnt so that mistakes are not repeated.
5.	Developers misunderstood that increased quality measures will also increase the project costs.	Developers know that improved quality increases the business profit.
6.	Quality was internally focused and understood that software has quality if it is just easy to maintain, reusable and flexible.	Quality is customer focused and it is described in terms of integrity, reliability, usability and accuracy.
7.	Quality task is conducted at project execution.	Quality task is conducted as project initiates.

10.2.2 PDCA Cycle

Q. Explain PDCA cycle in brief.

The most popular tool used to determine QA is the Shewhart Cycle, developed by Dr. W. Edwards Deming. This cycle consists of four steps: *Plan*, *Do*, *Check*, and *Act* i.e. PDCA.

1. **Plan :** Establish objectives and processes required to develop the proposed software product.
2. **Do :** Implement the planned process.
3. **Check :** Examine and evaluate the implemented process by testing the results against the predeter-mined objectives.
4. **Act :** Apply necessary actions for improvement if the result requires any changes.

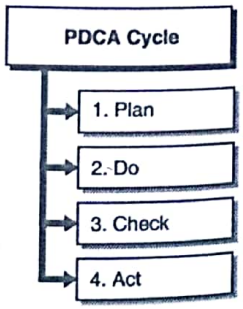


Fig. C10.1 : PDCA Cycle

10.3 SQA : Software Quality Assurance

- SQA serves as a customer's in-house representative that assists the software engineering team in achieving high-quality.
- SQA is considered as an *umbrella activity* that is applied throughout the SDLC process.
- SQA is a conformance to *explicitly stated software requirements* (functional requirements), *explicitly documented development standards, and implicit characteristics* (such as ease of use, maintainability, reliability, etc.) expected from the software.

10.3.1 Need of SQA

- SQA is a process that defines how software quality can be achieved and how the development organization knows that the software has the required level of quality.
- SQA group serves as the customer's in-house representative i.e. these people look at the software from customer's point of view.
- SQA monitors the software engineering processes and methods to ensure quality.
- SQA is a process of verifying or confirming that whether the products and services meet the customer expectations or not.
- SQA is a *process-driven approach* with specific steps to attain development goals. This process of QA considers design, development, production, and service.
- SQA group checks whether;
  - The software adequately meets the quality factors.
  - Software development has been conducted according to pre-established standards.
  - The technical disciplines (software engineers and testers) have properly performed their roles as part of the SQA activity.

Syllabus Topic : SQA Tasks

10.3.2 SQA Tasks

SQA is looked after by two different groups :

1. **Software engineers** who develop the software and have lot of technical knowledge.
   Software engineers address the quality by :
   - Applying effective technical methods and measures
   - Conducting formal technical reviews
   - Performing well-planned software testing
2. **Software quality assurance persons** who address the quality by :
   - Planning the quality assurance
   - Record keeping
   - Error reporting so as to improve the development process and prevent bugs from ever occurring.
   - Monitoring the development activities
   - Analysis and

The Software Engineering Institute (SEI) encompasses a set of SQA activities to evaluate the development process of the software product.

SQA activities

1. Prepare SQA plan for the project.

- The SQA plan is developed during project planning and after the requirement gathering.
- This SQA plan is then reviewed by all the stakeholders. Stakeholders are the persons who directly or indirectly interact with system such as customers, end-users, developers, testers, analysts, reviewers and etc.
- The SQA plan identifies :
  - Evaluations to be performed
  - Audits and reviews to be performed
  - Quality standards that are applicable to the project
  - Procedures for error reporting and tracking
- Documents to be produced by the SQA group
- Amount of feedback provided to the software project team

2. Participate in the development of the project's software process description.
   - The Software engineers select some process to develop the software and to help them, the SQA team reviews the process description to check:
     - The compliance of the development process with organizational policy
     - Internal software standards
     - Externally imposed standards
     - Other parts of the software project plan
3. Reviews the software engineering activities to verify it's compliance with the defined software process.
   The SQA team identifies, then documents and tracks the deviations from the process and verifies that the required corrections have been made.
4. Audits the designated software work products to verify their compliance with those defined as part of the software process.

SQA group audits

- The software work product
- Identifies, documents and tracks the deviations
- Verifies that the needed corrections have been made and
- Periodically reports the reviewed results of the software work product to the project manager.

5. Ensures that deviations in software work and work products are documented and handled according to a document procedure.
   These deviations may be encountered in the project plan, process description, applicable standard or technical work products.
6. Records any non-compliance and reports to senior management.
   The SQA team keeps track of the non compliance things until they are resolved.

Syllabus Topic : Goals

10.3.3 SQA Goals

Q. State the goals of SQA.

Goals of SQA activities in terms of Software Development

1. Assures that the software development will conform to the user requirements.
2. Assures that the software development will conform to planned schedules and budget.

3. Initiating and managing the activities to achieve greater efficiency of software development and SQA activities.

Goals of SQA activities in terms of Software Maintenance

1. Assures that the software maintenance activities will comply with the functional requirements.
2. Assures that the software maintenance activities will conform to planned schedules and budget.
3. Initiating and managing the activities to achieve greater efficiency of software maintenance and SQA activities

Syllabus Topic : Elements of SQA

10.3.4 Elements of SQA

- *Standards* and *Procedures* are the building blocks of SQA task; since; these provide the framework from which the software evolves.
- *Standards* are the pre-established criteria to which the *software products* are compared and *Procedures* are the established criteria to which the *software development* and *control processes* are compared.
- Therefore, standards and procedures both of them establish the prescribed methods for developing the software.
- Thus, proper documentation of standards and procedures is necessary to guide the SQA group in monitoring the process and evaluating the product.

Types of standards

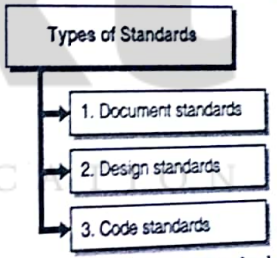


Fig. C10.2 : Types of Standards

1. **Document standards :** specifies the form and content for planning, controlling. The documentation provides consistency throughout the project. The NASA Data Item Descriptions (DIDs) provide the documentation standards.
2. **Design standards :** specifies the form and content of the design product. They provide rules and methods for translating the software requirements into the software design and then representing it in the design documentation.
3. **Code standards :** specifies the programming language in which the code is to be written and also define the restrictions on use of language features. It defines the legal language structure, style conventions, rules for data structure and interfaces and internal code documentation.
   - Procedures are explicitly stated criteria or steps that must be followed in carrying out a process.
   - All processes must have documented procedures.
   - Few processes in which procedures are required are configuration management, reporting the nonconformities and taking corresponding corrective actions, testing and also to conduct formal inspections.
   - Thus, in short, the SQA activities assure that both product and process comply with the established standards and procedures described in the QA portion of the management plan.

As we have already stated the ***standards*** are used for ***product evaluation*** and ***procedures*** are used for ***process monitoring***, we will see these both things in little detail.

Product evaluation

- It is an SQA activity that assures that quality standards regarding the product are being followed.
- The first products monitored by SQA group should be the project's standards and procedures.
- SQA assures that clear and achievable standards exist and thus these standards are achieved by the software product.
- Product evaluation assures that the software product reflects the requirements of the applicable standards as identified in the management plan.

Process Monitoring

- It is an SQA activity that assures that appropriate steps are undertaken to carry out the software development process.
- SQA monitors the process by comparing the actual steps carried out with those in the documented procedures.
- The quality assurance portion of the management plan specifies the methods to be used by the SQA process monitoring activity.

SQA Audit Activity

- This is also a building block of SQA because it is a fundamental SQA technique that looks at the process and product in depth and compares it with the established standards and procedures.
- Audits conduct the review of management, technical and assurance process to provide an appropriate quality and status if the software product.
- Audits assure that proper control procedures are being followed that include maintenance of documentation and accurate status reports of the process conducted.

Syllabus Topic : Metrics

10.4 Metrics for Software Quality

The following are the software quality factors and the needed metrics and measurements for each :

Fig. C10.3 : Software Quality Factors

1. Correctness

- A program must operate correctly else it provides little value to its users. Correctness is measured as the degree to which the software performs its required functionality.
- The most common measure for correctness is number of defects per KLOC where a defect is lack of conformance to requirements.

2. Maintainability

- Maintainability is the ease with which :
- An application can be corrected if an error is encountered

- A defect can be fixed
- A application is adapted in a changed environment
- A application is enhanced
- A simple time oriented metric is Mean Time to Change (MTTC) i.e. the time it takes to analyze the changed request, design an appropriate modification, implement the modification, test it and deliver it to the client. Fig. 10.4.1 which gives the scenario of online detection and offline repair. The measures MTBF (Mean Time Between Failure), MTTD (Mean Time To Detect) and MTTR (Mean Time To Repair) are the average times to failure, to detection and to repair.

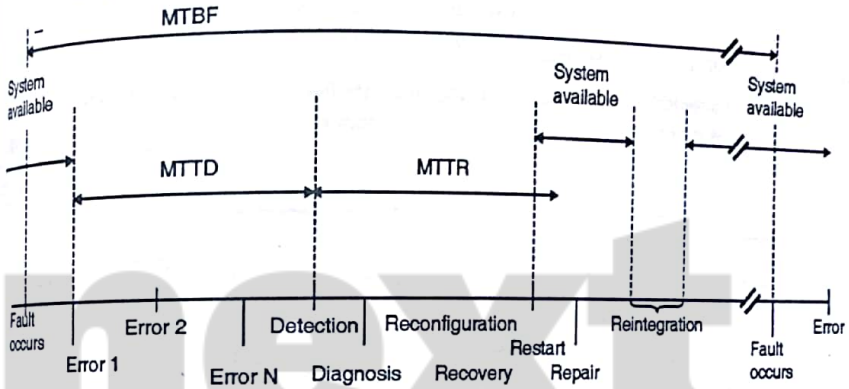


Fig. 10.4.1 : MTBF, MTTD and MTTR measures of maintainability

3. Integrity

- This attribute measures a system's ability to withstand attacks to its security. Attacks can be made on all three components of software: programs, data and documents.
- We cam measure integrity through following two additional attribute :
  1. Threat: It is the probability that an attack of a specific type will occur within a given time.
  2. Security: It is the probability that the attack of a specific type will be repelled. The integrity of a system van be then defined as :

  $$\text{Integrity} = \sum (1-(\text{threat} * (1-\text{security})))$$

4. Usability

- If a program is not easy to use, it is often meant to failure. Even if the functions that it performs are valuable.
- Usability is an attempt to quantify ease-of-use and can be measured in terms of characteristics.

Defect Removal Efficiency

- It is a quality metric that provides benefit at both the project and process level.
- DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.
- DRE is defined as: DRE = E/(E+D) where E is the number of errors found before delivery of the software to the end user and D is the number of defects found after delivery. The ideal value for

DRE is 1 i.e. no defects are found in the software. Generally, D is greater than 0 but DRE can be 1 when, as E increases the overall value of DRE begins to approach 1.

Other nine Classes of Measures

A. Product Measures

1. Errors, faults, failures
2. Mean-time-to-failure
3. Realibility growth and projection
4. Remaining products faults
5. Completeness and consistency
6. Complexity

B. Process Measures

1. Management control
2. Coverage
3. Risk, benefit, cost evaluation

C. Defect Amplification and Removal

- A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design and coding steps of the software engineering process.

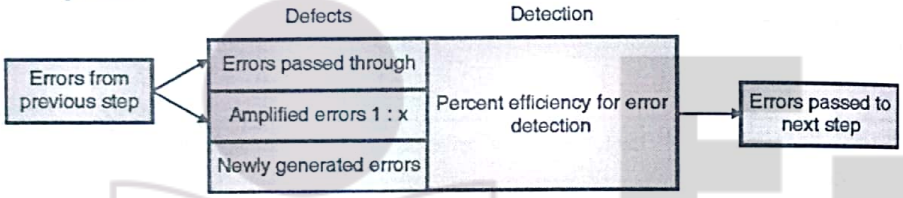


Fig. 10.4.2 : Defect Amplification model

- In case if the technical reviews fail to discover the newly generated defects, these may get amplified. The Fig. 10.4.2 represents the percent of efficiency in detecting errors.

Establishing a Software Metric Program

Following are the steps to conduct a goal-driven software metric program :

Step 1 : Identify your business goals
Step 2 : Identify what you want to know or learn
Step 3 : Identify your sub goals
Step 4 : Identify the entities and attributes related to your sub goals.
Step 5 : Formalize your measurement goals
Step 6 : Identify the quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
Step 7 : Identify the data elements that you will collect to construct the indicators that help answer your questions.
Step 8 : Define the measures to be used, and make these definitions operational.
Step 9 : Identify the actions that you will take to implement the measures.
Step 10 : Prepare a plan for implementing the measures.

10.5 Quality Factors

McCall's Quality Factors : Jim McCall produced the McCall's Quality Model in 1977 for the US Air Force to bridge the gap between users and developers. He tried to map the user view with the developer's priority.

McCall has categorized the software quality factors in to 3 categories:

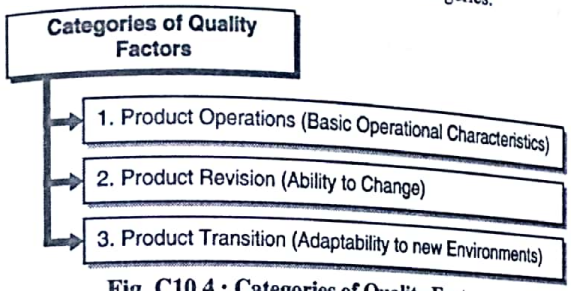


Fig. C10.4 : Categories of Quality Factors

1. **Product Operations (Basic Operational Characteristics) :** The quality criteria specified in this category talk about how easy it should be to handle and how efficiently it should use the resources and give bug free processing and expected outputs.
2. **Product Revision (Ability to Change) :** The quality criteria specified in this category say that the software should be easy enough to test, maintain and make any changes if required.
3. **Product Transition (Adaptability to new Environments) :** The quality criteria specified in this category says that it must be easy to transit (carry or load or install) the software on any platforms, should be able to share its code with the other languages on the platform and try to write the code i.e. reusable in future in any other software development.

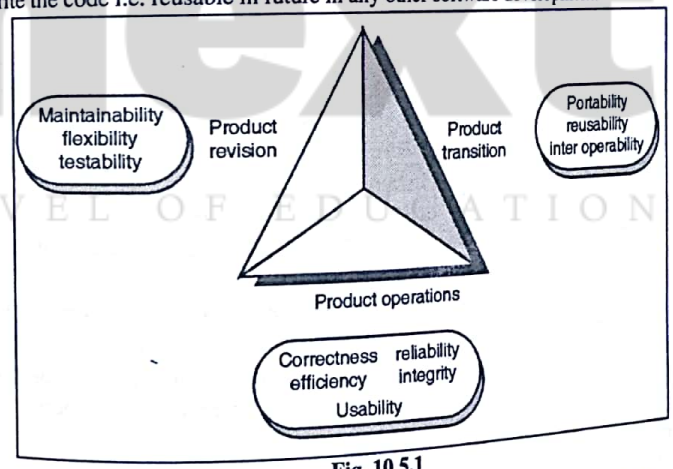


Fig. 10.5.1

The attributes of a quality/good software as shown in Fig. 10.5.1, categories are as below :

1. **Correctness :** Extent to which the functionality matches the specification
   It means the required functionality and correct results. Customer satisfaction depends on the degree to which customer requirements and expectations have been met. It should provide all the functionalities desired by the customer.
2. **Reliability :** Extent to which the system fails
   It means the *Bug free execution* i.e. assured performance, fully verified and validated processes, secured and safe to work when break down or power crisis. It should be reliable in all conditions. Reliability is measured in terms of the mean time between failures (average time between the

consecutive failures in a given period of life of a system), mean time to repair the equipment and mean time to recover (average time taken to return a system to the point where it failed and continue with its operation. Software reliability can be viewed from three different dimensions that specify the overall system reliability.

- *Hardware reliability* : It is the probability of hardware component failure and what time it takes to repair that component.
- *Software reliability* : It is the probability of the software failure i.e. how frequently software produces an unexpected and incorrect output. Software failures are different from hardware failures.
- *Operator reliability* : It is the probability of how frequently the operator of a system will make an error.

3. **Efficiency :** System resource (including CPU, disk, memory, network) usage.

It means the efficient use of resources. Software is said to be efficient if it uses all its resources i.e. memory, processor and storage in an effective manner. The software design and architecture should be such that it gives you the response in the least processing time, using the resources in the best possible way.

4. **Integrity :** Protection from unauthorized access

Integrity is related with the extent of access to software by unauthorized persons can be controlled.

5. **Usability :** Ease of use.
   - It means that the software must be user friendly. The software should have a good documentation and user manual which may include the installation and the process of using the software. This makes easy for the new to learn and operate just by studying the manual.
   - It acts as a bridge between the user requirements and the developed system. Usability counts in the terms of effectiveness, efficiency and satisfaction.
6. **Maintainability :** Ability to find and fix a defect.
   - For all the changes desired by the customer or the user, the software engineer has to respond fast. And this is possible only if the software design and its architecture are so chosen that changes can be carried out in the shortest time, without affecting the overall integrity of the software.
   - The change could be to correct the mistakes, expand its scope or adapt to new technology.
7. **Flexibility :** Ability to make changes required as dictated by the business.

The software should be developed so that if user demands any changes in the system at coding or testing phase i.e. in the middle of the SDLC, then it should be easy to insert these changes in the existing modules.

8. **Testability :** Ability to validate the software requirements.

It should need less effort to test a program so that it performs its intended function. As the complexity of the program increases, the efforts to test the software also increase. Testability can be defined in terms of the following attributes that try to uncover maximum number of defects with minimum efforts.

(i) **Operability :** The better the software works, the more efficiently it can be tested

(ii) **Observe-ability :** It is based on the fact that 'what you see is what you get' (WYSIWYG)

(iii) **Controllability :** The better way to control the software quality is to automate and optimize the tests.

(iv) **Decomposability :** isolating the problems and performing smarter re-testing.

(v) **Simplicity :** less there is to test, the more quickly it can be tested.

(vi) **Stability :** Fewer the changes, fewer the disruptions to testing.

(vii) **Understandability :** The more information we have about the system, the smarter we can test

9. **Portability :** Ability to transfer the software from one environment to another.

It is about developing the software to run on one operating system or hardware configuration while being conscious of how it might be shaped with minimum effort to run on other operating systems and hardware configurations as well.

10. **Reusability :** Ease of using existing software components in a different context.

It gives the concept of 'Write once and Use many times!' For example, writing functions or sub procedures to receive variable parameters. The calling code passes the values to the parameters and the called procedure processes them as needed. The advantage of this approach is that – once the code is written and fully tested, it can be used lot number of times anywhere.

11. **Interoperability :** Ease to which software components work together.

The software development should be so that it can interact with other products. For example, Word processor can incorporate charts from Ms-Excel or data from databases. It deals with the interface between software products over a communication network.

12. **Cost-effective :**

The development of the software within the cost and budget depends on efficient design and a high level of project management effort.

Syllabus Topic : Formal Approaches to SQA

10.6 Formal Approaches to SQA

Three formal approaches to SQA are :

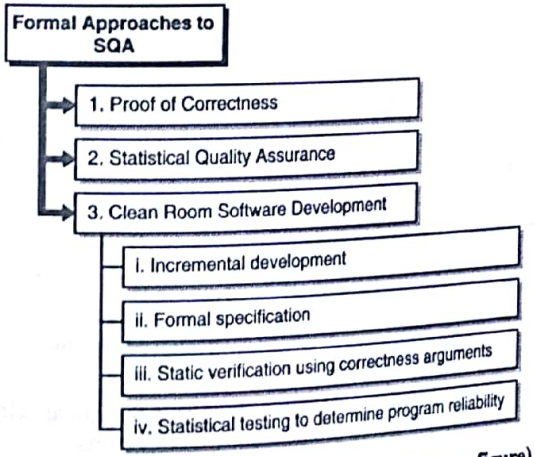


Fig. C10.5 : Formal Approaches to SQA (Change figure)

10.6.1 Proof of Correctness

The aim of this approach is to prove a program correct.

- It is a mathematical proof that when executed by computer program yields correct results.
- Hypothesis : it is a condition that the program must satisfy before the program is executed. This condition is also called as 'pre-condition'.
- Thesis : it is a condition that the program must satisfy after the execution of the program. This condition is also called as 'post-condition'.

10.6.2 Statistical Quality Assurance

- Identify software defects
- Each defect is tracked back to its cause
- correct the causes behind the defects.

10.6.3 Clean Room Software Development

- The name 'Clean Room Software Development' is derived from the 'Cleanroom' process of semiconductor fabrication.
- The philosophy behind this technique is *defect avoidance* rather than defect removal.
- Clean room software development is an engineering process to develop a high-quality software with certified reliability.
- As part of IBM's Federal Defence System, Harlan Mills, Dyer and Linger developed the CSE methodology in the early 1980s.
- Their followed the motto 'prevention is better than care' and thus, aimed to prevent software errors before they happen rather than correcting them after they occur.
- The four main components of clean room approach : to CSE are :

i. **Incremental development :** In clean-room process, each increment is developed separately and tested in a simulated environment to check the quality of the subsystem. The results of previous increments can be used to improve the quality of next increments which makes the process more successful. Incremental, prototyping and spiral SDLC models are useful here to manage the risks.

Advantages

- Allows early and continuous quality assessment throughout the development process.
- Provides increased user feedback at each and every step of development.
- Repairs if any process related problems occur.
- Allows the users to change their requirements

ii. **Formal specification :** The Box Structure Method is used for specification and design of the CSE process.

iii. **Static verification using correctness arguments :** Verification of program correctness is done by the team review based on correctness questions. Mathematical verification techniques are used to verify the program correctness.

iv. **Statistical testing to determine program reliability :** Statistical principles are used to test the software and certify its reliability. The sequence of clean room tasks for each increment is described in Fig. 10.6.1

Task 1 : Requirement gathering

A more detailed and descriptive user requirements are gathered and sustomer-level specification is developed.

Task 2 : Box structure specification

It makes use of box structures to describe the functional specification that confirms to the operational analysis principles. The box structures isolate and separate the creative definition of behaviour, data and procedures at each level of refinement.

Task 3 : Formal design

Clean-room uses box structure specification for specification and design of the CSE process. The box structure makes the formal design in three levels of abstractions :

1. *behavioral view* : represented by black box that describes the interaction of the whole system with the application environment.
2. *finite state machine view* : represented by state box
3. *procedural view* : represented by clear box that describes the procedures in the system.

Task 4 : Correctness verification

A series of rigorous correctness verification activities are conducted on the design and then on the code. Verification begins by verifying the highest level box structure i.e. the black box verification and then moves towards the design details i.e. state box and then towards the code i.e. clear box verification. This technique finds the errors faster and reduces the development and testing costs. This technique replaces the unit testing because the errors are found before testing.

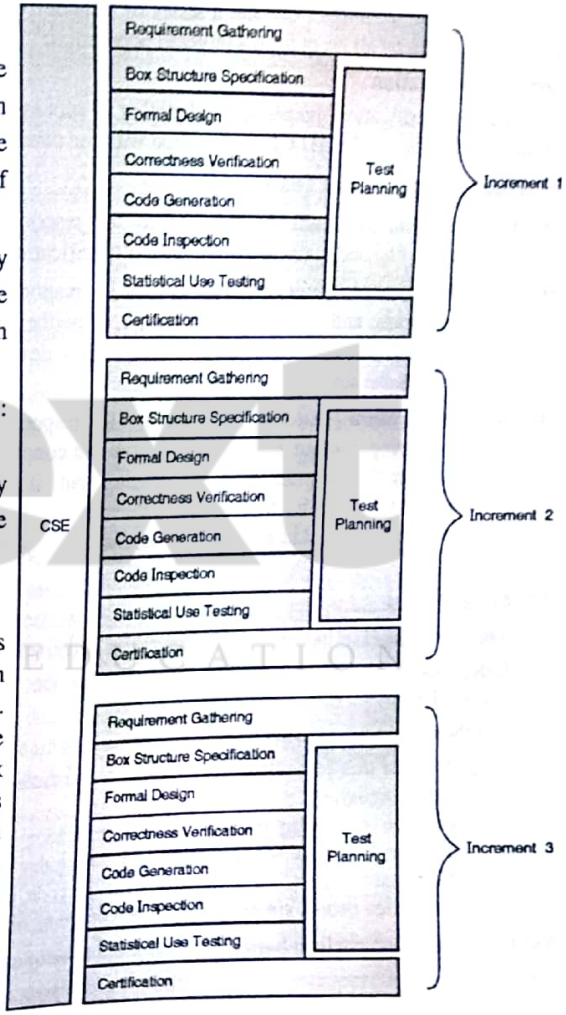


Fig. 10.6.1 : Clean-room Software Engineering (CSE) process

Task 5 : Code generation

The box structure specification represented in a specialized language is translated into appropriate programming language.

Task 6 : Code inspection

Standard Walkthroughs and inspections are conducted to ensure semantic and syntactic conformance of the code.

Task 7 : Test planning

A suite of test cases that exercise a probability distribution of usage is planned and designed. Test suits are developed in parallel with specification, verification and code generation.

Task 8 : Statistical use testing

Statistical use techniques execute a series of tests derived from a statistical sample of all possible program execution by all users from a targeted population.

Task 9 : Certification

Once the verification, inspection and testing is successfully completed, that particular increment (module) is certified as READY for integration with the other increments (or modules or components).

Cleanroom process Team

1. **Specification team :** This group is responsible for developing and maintaining the system specification.
2. **Development team :** This group is responsible for developing and verifying the software. It neither executes the software nor does the compilation. It only develops and inspects the software.
3. **Certification team :** This group is responsible for developing a set of statistical test cases and conducts these tests on the software after the development. It also uses reliability growth models to determine the software reliability and if it is acceptable, it certifies the increment.

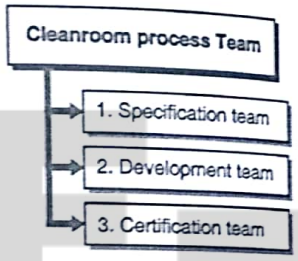


Fig. C10.6 : Cleanroom process Team

Advantages

- The results of CSE have been very impressive in delivered systems.
- Independent incremental assessment shows that the process is no more expensive than other approaches.
- But then, the process is not widely used because of the following reasons:
  - A belief that cleanroom methodology is too theoretical, too mathematical and too radical for use in reality.
  - Replace unit testing with correctness verification and statistical quality control that represents a major difference with the techniques used today in software development.
  - No constraints are set on how to write the code, code review is done mentally and verbally, no written proofs are required and no compiling of code is done.

Table 10.6.1 : Comparison between Traditional development models and Cleanroom development

	Typical Development	Cleanroom Development
1.	Specification is usually incomplete for external behavior.	Specification is complete providing the precise and complete description for external behavior
2.	From specification, code is informal, debugging is done to verify.	Box Structures are used to refine and verify the specification and designs.
3.	Failures are common and accepted	Failures are not at all accepted.
4.	It attempt to perform code coverage test but poor reliability prediction.	It is usage model based and predicts the field reliability

Syllabus Topic : Six Sigma

10.7 Six Sigma

- *Sigma* is the Greek letter that represents standard deviation in statistics. *Six Sigma* generated from 'sigma' has the same idea behind it i.e. it relies heavily on statistical techniques to reduce defects and measure quality.
- Six sigma stands for six standard deviations that strives for perfection – allows only 3.4 defects per million opportunities for each product or service transaction.
- Six-sigma was first started by Motorola in 1980's in its manufacturing division where millions of parts are made using the same process repeatedly.
- Slowly, Six Sigma was being used in other non manufacturing processes. Today, Six Sigma is applied to many fields such as Services, Medical and Insurance Procedures, Call Center, etc.
- Six-Sigma methodology involves the techniques and tools needed to improve the capability of the development process and thus, reduce the defects in it. Therefore, Six-Sigma is the method of constantly reviewing and re-tuning the process.
- Six Sigma experts (Green Belts and Black Belts) can either evaluate the existing business process or determine the ways to improve the existing process or design a brand new business process using *DFSS* (*Design For Six Sigma*) principles. Usually, it is easier to design a new process than refining an existing process.

Six-Sigma methodology defines **three core steps** abbreviated as **DMA :**

1. Define the customer requirements, deliverables and project goals via well defined methods of customer communications.
2. Measure the performance of the existing process and its output to determine the current quality performance.
3. Analyze the defect metrics and determine the vital few causes.

If Six Sigma methodology is used in ***improving the existing software process***, it suggests two additional steps with the above three core steps that can in all be abbreviated as DMAIC

1. Improve the process by eliminating the root causes of defects.
2. Control the process to ensure that future work doesn't re-introduce the causes of further defects.

If Six Sigma methodology is used to ***create a brand new software process*** from ground up, it suggests two other additional steps to the three core steps that can in all be abbreviated as

DMADV

1. Design the process :
   - To avoid the root causes of defects
   - To meet the customer requirements
2. Verify that the process model will in fact avoid the defects and meet customer requirements.

The Six-Sigma methodology leads to defect reduction and improvement in profits, product quality and customer satisfaction.

The three key elements of Six-Sigma are :

1. Customer Satisfaction.
2. Defining Processes and defining Metrics and Measures for Processes.

3. Team Building and Involving Employees - The company involves all employees and provides opportunities and incentives for employees to focus their talents and ability to satisfy customers.

Syllabus Topic : Software Reliability

10.8 Software Reliability

Reliability means the probability of failure-free operation of software in a specified hardware environment for a specified time.

It is the probability of the software failure i.e. how frequently software produces an unexpected and incorrect output. Software failures are different from hardware failures.

One of distinct characteristics of reliability is that it is objective, measurable, and can be estimated else many of the software quality attributes are subjective. These measurable criteria are called as *software metrics.*

☞ **Need**

With increasing number of software and expanding scope, software is in boom and simultaneously causes lot of failures. And the causes for these failures are the poorly designed GUIs and program coding.

10.8.1 Reliability Measures

- Reliability metrics are the units used to measure the system reliability
- **Software reliability** can be measured by counting the number of operational failures and relating these to the demands made on the system at the time of failure.
  - Probability of Failure on Demand (POFOD) = 0.001
- Where, for one in every 1000 requests the service fails per time unit of operation.
  - Rate of Fault Occurrence (ROCOF) = 0.02
- Where, two failures for each 100 operational time units of operation
- ***Software Reliability*** can be measured in terms of Mean-Time-Between-Failures (MTBF) where;

  Reliability = MTBF = MTTF + MTTR ;

  Where; MTTF is mean-time-to-failure and MTTR is mean-time-to-repair respectively.
- Measuring MTBF is more useful than measuring number of defects/KLOC because each defect only provides very little indication of the reliability of a system.
- ***Software availability*** is also a quality attribute which is described as the probability that a program is operating according to requirements at a given point in time. It is measured as;

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

☞ **Reliability Measurement Process**

The process of measuring reliability is illustrated in Fig. 10.8.1

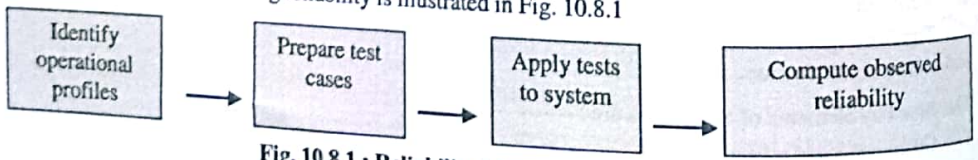


Fig. 10.8.1 : Reliability Measurement Process

Step 1 : Study the existing systems of the same type to identify the operational profile.

Step 2 : Then design a set of test cases that reflect the identified operational profile. These test cases can be generated from studying the existing system as stated in the above step using the test data generator tool which is a Computer Aided Software Testing (CAST) tool.

Step 3 : Apply the tests to the system and count the number and type of failures that occur.

Step 4 : After statistically computing the number of failures, you can compute the software reliability and then find the appropriate reliability metric value. This, this process is also called as statistical testing which aims to measure the software reliability.

10.8.2 Reliability Models

Software reliability models are useful in describing the characteristics of how and why software fails also quantifies the software reliability.

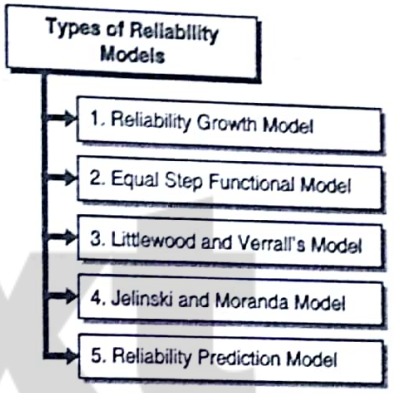


Fig. C10.7 : Types of Reliability Models

➔ **1. Reliability Growth Model**

- This model describes how the system reliability changes over time.
- During the testing process, as failures are discovered; the faults causing these failures are fixed so as to improve the software reliability i.e. changing of reliability from poor to better.

➔ **2. Equal Step Functional Model**

- This model illustrates the concept of reliability growth that describes that the reliability goes on increasing each time a fault is discovered and repaired and then a new version of software is created.
- This model assumes that the software repairs are always correctly implemented so as to reduce the number of software faults and associated failures in each new version of the system. All this description is signified in Fig.10.8.2

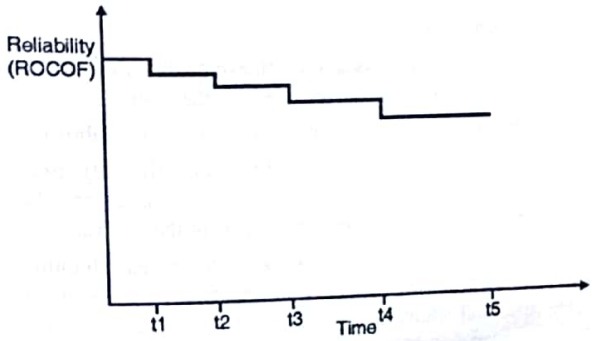


Fig. 10.8.2 : Equal Step Function Model representing Reliability Growth

- In practice, debugging doesn't always fix the software faults because changing a program sometimes introduces new faults into it. The probability of occurrence of these faults may be higher than the occurrence of the faults that have been repaired.
- Repairing the most common faults rather than repairing the occasionally occurring faults improves the reliability growth comparatively.

3. Littlewood and Verrall's Model (Random Step Function Model)

- This model considers the problem by introducing a random element in the reliability growth improvement effected by a software repair. Therefore, each repair does not result in an equal reliability improvement but varies on the basis of *random* perturbation.
- This model allows a negative growth when a software repair produces further errors thus decreasing the reliability. The reason behind this is that the frequently occurring faults are likely to be discovered early in the testing process.

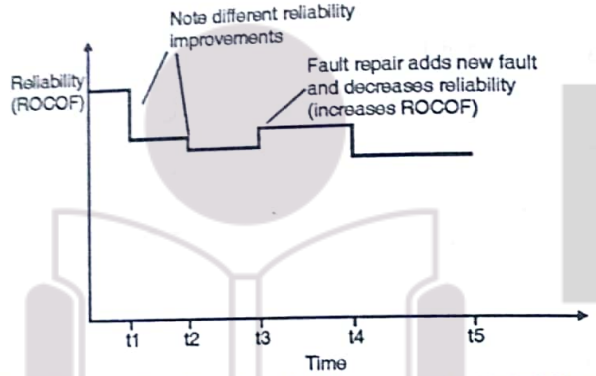


Fig. 10.8.3 : Random Step Function Model representing Reliability Growth

4. Jelinski and Moranda Model

- This model states that each time an error is repaired the reliability does not increase by a constant amount.
- Rather the reliability growth is by assuming that fixing of an error is proportional to the number of errors present in the system at that time.

5. Reliability Prediction Model

- Testing is very expensive but important to achieve quality software. However, there is a need to stop testing as soon as possible and not over-test the system.
- Testing must be stopped when the required level of system reliability is achieved.
- This required level of reliability can be checked using reliability prediction model and if this model predicts that the required level of reliability will never be achieved, then in this case, the manager must decide to rewrite the components of the software.
- The reliability can be predicted by comparing the measured reliability value with the known reliability model. That means, reliability is predicted by matching the achieved reliability value to some historical reliability data. You can extrapolate the model to the required level of reliability and observe at what point is the required level of reliability is achieved.

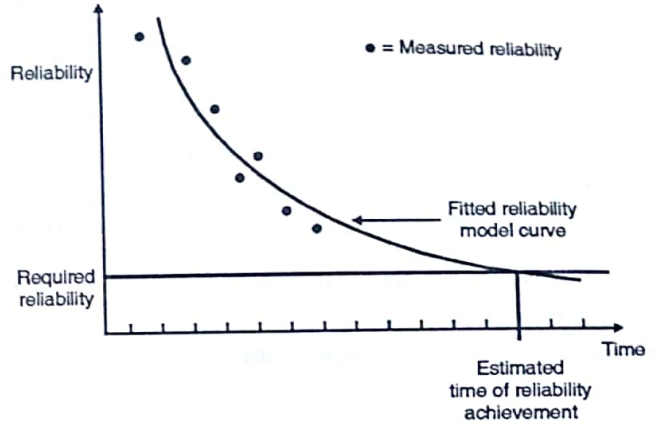


Fig. 10.8.4 : Reliability Prediction Model

Predicting the system reliability has two main benefits

1. **Planning of testing :** Using the testing schedule, we can predict the completion time of testing. And if this prediction says that the completion time is after the planned delivery date, then you have to use additional resources to conduct testing and debugging so as to accelerate the rate of reliability growth.
2. **Customer negotiations :** Sometimes the reliability model shows that reliability growth is very slow and testing is also giving relatively very less benefit, or if the model predicts that the required reliability level will never be reached, then in these cases, it would be worth to renegotiate the reliability requirements with the customer of the system.

10.9 SQA Planning and Standards

- SQA planning is the process of developing a quality plan that lists the desired software qualities and describes how these are to be verified.
- SQA plan involves those organizational standards that are appropriate to a particular product and development process.

Quality plan includes

1. **Proposed product introduction :** It provides detail description of the product, its intended market and quality expectations from the product.
2. **Product plans :** It includes the critical release dates and responsibilities for the product along with plans for distribution and product servicing.
3. **Process description :** It describes the development and service processes that will be used for product development and management.
4. **Quality tools :** It describes the identification and justification of critical product quality attributes.
5. **Risk and risk management :** It describes the most probable risks that may affect the product quality and the actions to address these risks.

The overview of the quality plan is as described above but it differs in describing the details depending on the size and type of the system that is being developed. But the plan should be designed as short as possible because if it is too long then people will not read it and the purpose of preparing the quality plan is ultimately lost.

☞ **The SQA plan**

- The SQA plan provides a road map for monitoring the software quality assurance.
- The SQA plan is developed by the SQA group.
- The SQA plan serves as a template for conducting the SQA activities that are assigned for each software project.
- The IEEE has published a standard format for SQA plan. This plan has the following structure.

1. **Purpose and Scope of the plan**
2. **References** i.e. which existing system or documents are studied to get the knowledge about the new system.
3. **Management** describes the roles and responsibilities of SQA in the organizational structure, tasks of SQA group, their roles and responsibilities related to product quality.
4. **Documentation** describes all software engineering work products produced as part of the software development process. It includes project documents, design models, technical documents and user manuals.
5. All applicable **Standards, Practices, and Conventions** list the standards/practices applied during the software development process.
6. **Reviews and Audits** provide a brief overview of the review and audit approach to be conducted during the project development.
7. **Tests** describe the test plan and procedures and the record keeping requirements.
8. **Problem reporting and corrective actions** define the appropriate procedures required for reporting, tracking and ultimately resolving the errors.
9. **Tools and Methods** that describe the assembling, safeguarding and maintaining of all SQA related records.

10.9.1 SQA Standards

SQA defines a set of standards that should be applied to the software development process or software product.

QA establishes two types of standards :

1. **Product standards :** These set of standards are applied to the software product that is being developed. They include the document standards such as the structure of requirements, standards commitment header from an object class definition and coding standards that define how a programming language should be used.
2. **Process standards :** These set of standards are applied to the software process that are followed while developing the software. They include the definitions of specifications, design and validation processes and a description of the documents that should be written in the course of these processes.

The product and process standards are inter-linked with each other – product standards are applied to the output of the software process and process standards include the software process activities that ensure that product standards are followed.

☞ **Need of SQA standards**

1. Provide the knowledge about the best appropriate practices followed in the company which is often achieved after a great deal of trial and error work. All these best practices are built up as a standard which guides the company from avoiding repeating the past mistakes.
2. Standards provide a framework for enforcing quality assurance activities as they inculcate bet practices ensuring that the standards have been properly followed.
3. They help in continuity where work carried out by one person can be handed over and continued by other person.
4. Standards ensure that all the engineers within an organization adapt the same practices. Therefore, the effort needed to learn new practices is reduced.

☞ **Setting steps of SQA standards**

Quality managers who set the standards of the organization have to be well-equipped with skills and resources and should follow the below steps to build the standards;

1. Involve the software engineers in the selection of the product and process standards. Engineers should understand the importance of standards, why they have been designed and must be committed to these standards. The standards document should not just state the standards but they should also mention why particular standardization decisions have been made and what is its use.
2. Review and modify the standards periodically to cope with the changing technologies. Once the standards are developed, they need to be displayed in the organization's standards hand book and must be periodically updated to reflect the changing requirements and technologies.
3. Provide the software tools to support the standards whenever possible. Using the manual development process makes it difficult to comply with the standards but the use of tools provides comparatively more compliance and also reduces the extra efforts needed in that process.

10.9.2 ISO Quality Standards

- There are various international set of standards set out by International Standards Organization (ISO) that can be applied in the development of a quality management system across all industries.
- These ISO standards are concerned with the quality process in organizations that design, develop and maintain the products.
- One of such ISO standards is ISO 9126 that is specially aimed to software development and sets out some key quality attributes to assure the quality of the product being developed.

Syllabus Topic : The ISO 9000 Quality Standards

10.9.2(A) ISO 9000 Quality Standard

ISO 9000 is a set of QA standards that define a basic set of good practices to guide the company in delivering the user satisfying product.

☞ **Benefits of ISO 9000**

1. It focuses on the development process and not the product. It is concerned about the way an organization goes about its work and not the results of the work. And this is because having a quality development process will ultimately help in achieving quality product.
2. It only illustrates the process requirements and not how they are to be achieved. For example, the ISO 9000 standards says that a software team should plan and perform product design reviews but it doesn't say how that requirements should be accomplished.

The sections of ISO 9000 that deal with software are ISO 9001 and ISO 9000-3.

- The ISO 9001 is for business that designs, develops, installs, and services the products.
- The ISO 9000-3 is for business that develops, supplies, installs, and maintains the computer software.

10.9.2(B) ISO 9001

- It describes various aspects of the quality process and lays out the organizational standards and procedures that company should define.
- These should be *documented* in an organizational quality manual which should include the descriptions of the defined processes that have been followed during product development.

Documentation standards in ISO 9001

Documentation standards are important because documents are the only tangible way of representing the software and the software process. There are three major types of documentation standards :

1. **Document process standards :** These standards define the process that should be followed for document production.
2. **Document standards :** These standards govern the structure and presentation of documents.
3. **Document interchange standards :** These ensure that all electronic copies of documents are compatible.

The Fig. 10.9.1 depicts the model of a documentation process where - drafting, checking, revising and redrafting are iterative processes. It should continue until the document of acceptable quality is produced. The acceptable quality level depends on the document type and the potential readers of the document.

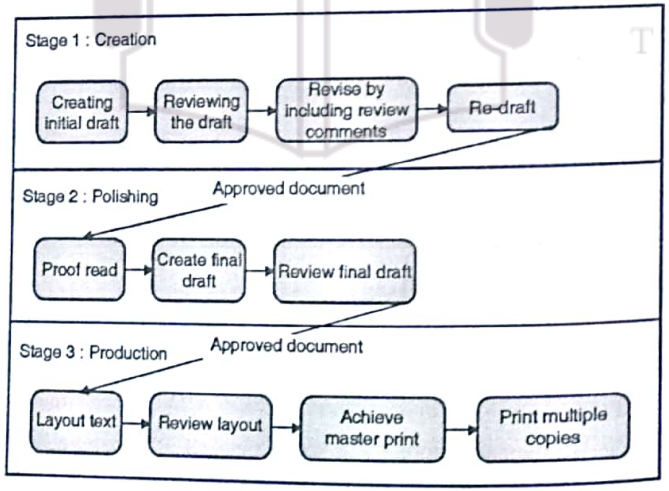


Fig. 10.9.1 : Documentation Procedure

Few Documentation Structures that may be developed are;

- **Document identification standards :** Thousands of documents are developed in large projects where each needs to be identified uniquely. This identifier is usually defined y the configuration manager.
- **Document structure standards :** These define the conventions used for page numbering, page header and footer information and section and sub section numbering.
- **Document presentation standards :** These include the definition on fonts, and styles used in the documentation, the logos, the company name and the colour opted to highlight the document structure.
- **Document update standards :** It reflects the changes in the system i.e. it provides a consistent way of indicting the document changes.

Criteria set by ISO 9000-3 to create a better software development process :

- Develop detailed quality plan and procedures to control configuration management, verification and validation, bug fixing and corrective actions.
- Prepare and receive approval for a software development plan that includes a definition of the project, a list of the project's objectives, a project schedule, a product specification, a description of how the project is organized, a discussion of risks and assumptions and strategies for controlling it.
- Communicate the specification to customer in the language that makes them understand and helps them in validating during testing process.
- Plan, develop, document and perform software design review procedures.
- Develop and document software test plans
- Develop methods to test whether the software meets the user requirements.
- Perform software validation and acceptance tests.
- Maintain the test cases and test results.
- Prove that the product is ready before it is released.
- Use statistical techniques to analyze the software development process and to evaluate product quality.

10.9.2(C) ISO 9126 Quality Standard

The ISO 9126 standard identifies 6 key quality attributes.

- **Functionality** : Degree to which software satisfies stated needs.
- **Reliability** : The degree to which software is up and running in unfavourable conditions.
- **Usability** : The degree to which software is easy to use.
- **Efficiency** : The degree to which software makes optimum use of resources.
- **Maintainability** : The ease with which the software can be modified.
- **Portability** : The ease with which software works on different environments.

Few other Quality Standards are :

- TQM (Total Quality Management)
- SEI CMM (Software Engineering Institute set of Capability Maturity Models)

ISO 15504

Syllabus Topic : Capability Maturity Model

10.10 CMMI Process Improvement Framework

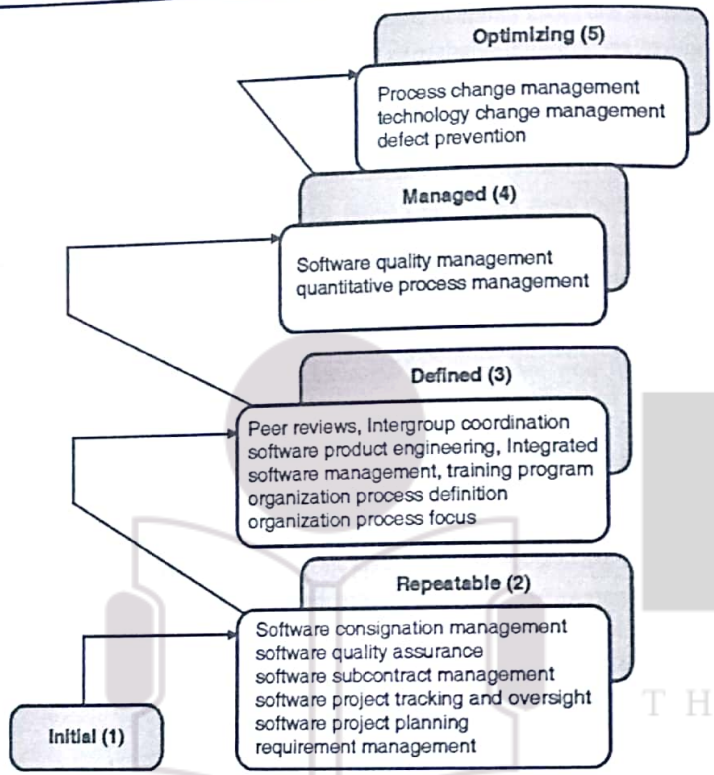


Fig. 10.10.1 : CMMI and Key Process Areas

- *Capability Maturity Model (CMM) is a maturity model applied within the context of Software Project Improvement framework.*
- *CMM is a specific* approach taken for quality assurance.
- The CMMI (Capability Maturity Model Integration) is used to implement Quality Assurance in an organization.
- The CMMI describes an evolutionary improvement path from an adhoc, immature process to a mature, disciplined process which describe the key elements of an effective software process.
- The CMMI includes key practices for planning, engineering, and managing the software development and maintenance which helps in improving the ability of organizations to meet goals for the cost, schedule, functionality, and product quality.
- The CMMI helps in judging the maturity of an organization's software development process and compares it to the state of practice of the industry.
- The CMMI categorizes five levels of process maturity :

  Level 1-Initial : The software process is characterized as ad hoc and chaotic. The success generally depends upon individual efforts.

Level 2-Repeatable : Basic project management processes are established to keep track of cost, schedule, and functionality. Then, the same process is used repeatedly for all the similar type of projects.

Level 3-Defined : The software development process is documented, and integrated into an approved standard software development process for the organization. All projects use this approved version of the organization's standard software process for developing and maintaining the software.

Level 4-Managed : Detailed measures for software process and product quality are collected, understood and implemented.

Level 5-Optimizing : Continuous process improvement is done by continuous feedback from the process and inculcating innovative ideas and technologies.

At this level, the entire organization focuses on continuous Quantitative feedback from previous projects which is used to improve the project management.

The software process at this level can be characterized as continuously improving the process performance of their projects.

Improvement is done both by incremental enhancements in the existing process and by innovations using new technologies and methods.

These levels are decomposed into several **key process areas.**

- **Process Change Management :** To identify the causes of defects and prevent them from re-occurring.
- **Technology Change Management :** To identify beneficial new technologies and incorporate them in an orderly manner.
- **Defect Prevention :** To continuously improve the process inorder to improve the quality productivity and thus decrease development schedule and cost.

Drawback of CMM

- The *CMM* does not describe how to create an effective software development process.
- The qualities it measures are practically difficult to implement in an organization.

Review Questions

Q. 1 Why is software quality important ?
Q. 2 What are the various causes behind the errors in a software product ?
Q. 3 State the difference between quality control and quality assurance.
Q. 4 Explain PDCA cycle in brief.
Q. 5 State the goals of SQA.
Q. 6 State and explain the quality metrics and measurements.
Q. 7 List down the McCalls quality factors.
Q. 8 What are the three formal approaches to SQA ? Explain cleanroom process in detail.
Q. 9 Explain six-sigma in detail.
Q. 10 Explain the software reliability measures in brief.
Q. 11 Write in short about various ISO quality standards.
Q. 12 Explain CMMI framework in detail.

UNIT III

CHAPTER 11

Software Testing

Syllabus

Verification and Validation, Introduction to Testing, Testing Principles, Testing Objectives, Test Oracles, Levels of Testing, White-Box Testing/Structural Testing, Functional/Black-Box Testing, Test Plan, Test-Case Design

Syllabus Topic : Verification and Validation

11.1 Verification and Validation

V & V stands for Verification and Validation. It is a whole life-cycle process - it is applied at each stage of software development process.

- **Verification**
  - "Are we building the product right ?"
  - Software should conform to its specification (SRS). It is about confirming with the user specified requirements.
- **Validation**
  - "Are we building the right product ?"
  - Software should do what the user 'really requires'. It is about confirming with the proper functionality and performance of the system with good security of data.

Table 11.1.1 : Difference between Verification and Validation

Sr. No.	Verification	Validation
1.	It is the process of confirming whether the software meets its requirement specification.	It is the process of confirming whether the software meets user requirements.
2.	Inspections, walkthroughs and reviews are the examples of Verification.	Structural testing, black box testing, integration testing, system testing, acceptance testing are the examples of validation.

Sr. No.	Verification	Validation
3.	It is usually a process of static testing i.e. inspecting without executing on computer.	It is usually a process of dynamic testing i.e. testing by executing on computer.
4.	It is the process of confirming if the phases are completed correctly.	It is the process of determining if product as a whole satisfies the requirements.
5.	It is the process of examining the product to discover its defects.	It is the process of executing a product to expose its defects.
6.	It answers, "Are we building the product right"?	It answers, "Are we building the right product"?

- **V & V Goals**
  - To establish confidence that the software system is 'fit for purpose'. This does NOT mean system with zero defects. It ensures that system is good enough for its intended purpose.
- **This level of degree of confidence depends on**
  - ***Software's functions*** i.e. how useful the software is to an organisation.
  - ***User expectations*** usually users have low expectations from the software and they are not at all surprised if the system fails during the use.
  - ***Marketing environment*** While marketing the software product, the sellers must also consider the competing software, the price that the customers are willing to pay for the software and the required schedule for delivering the system.
- **Objectives of V&V Process**
  - Discover the defects in a system;
  - Assess whether or not the system is useful.
- **Advantages of V and V Process**
  - Prevents faults and stops fault multiplication.
  - Avoids the downward flow of errors.
  - Earlier detection of errors leads to Lower defect Resolution cost.
  - Improves quality and reliability.
  - Reduces the amount of Re-work.
  - Improves Risk Management
  - Allows testers to be active in the early phases of the project's lifecycle.

11.1.1 Static and Dynamic Verification

Q. Differentiate: Static and dynamic testing

Verification and Validation of software can be done by two methods - Static and Dynamic.

Table 11.1.2 : Difference between Static and Dynamic Verification

Sr. No.	Static Verification	Dynamic Verification
1.	It is about reviewing the software development documentation. It doesn't include any inputs and outputs i.e. - No verification of actual outputs against the expected outputs. This is same as Software inspections.	It is not reviewing. It is just opposite to Static verification. Dynamic Verification is about verifying whether expected and actual outputs match or not. Dynamic Verification allows the developer or user to give any type of inputs and cross checks whether the actual output of these inputs match with the expected outputs. This is same as Software Testing.
2.	It is *about prevention*	It is *about cure*.
3.	More cost-effective than dynamic testing	Less cost-effective
4.	Achieves 100% statement coverage in short time.	Achieves less than 50% statement coverage because dynamic testing finds bugs only in the executable code.
5.	Takes less time to verify as it involves only checking the documents - it doesn't involve any inputs or executions.	Dynamic testing may involve running several test cases, each of which may take comparatively longer time.
6.	It can be done before compilation	It can take place only after compilation and linking
7.	Static verification can finds syntax errors, hard code i.e. hard to maintain and test, code that does not conform to coding standards.	Dynamic testing finds fewer bugs than static testing as most of them are detected in static verification.
8.	Example : Software inspections. This is concerned with analysis of the static system representation to discover problems.	Example : Software testing. Concerned with exercising and observing product behaviour.

11.1.2 V and V Model

The below Verification and Validation model (V & V Model) defines various levels V & V activities that are under taken at different stages of SDLC phases.

- V and V is a whole SDLC life-cycle process – it is applied at each phase of software development life cycle process.
- System verification and validation is started early in the development process.
- Careful V and V planning is must to get the most out of testing and inspection processes.
- The V and V plan should identify the balance between static verification and testing (dynamic verification).
- The below Verification and Validation model (V and V Model) defines various levels of V and V activities that are under taken at different stages of SDLC phases.
- You will see that this V and V Model looks like **'V'** shaped and so it is also called as 'V-model'.

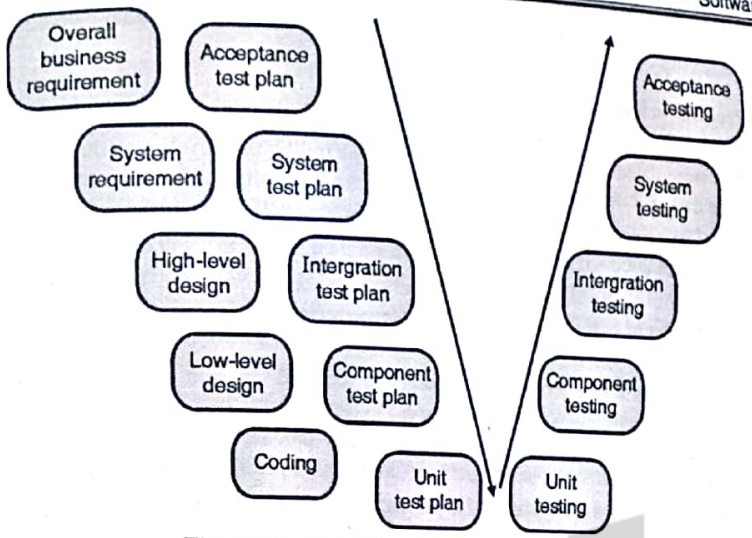


Fig. 11.1.1 : V & V Model / Levels of Testing

V-model is split into two parts :

1. Test design (on left side of V) is done early
2. Test execution (right side of V) is done in the end

By performing early design of tests and deferring only the test execution till the end will achieve three important advantages:

1. Achieves more parallelism and reduces the end-of-cycle time taken for testing.
2. By building the test design for each activity of the system, we build better upfront validation. Thus, this reduces the last-minute errors.
3. Tests are designed by the people with appropriate skill tests.

Syllabus Topic : Introduction to Testing

11.2 Introduction to Testing

- **Testing** is done by software tester to find **bugs** in a system as early as possible and notify the same to developers so that they get fixed.
- In 90's, Testing was not given that important and thus, systems developed then had lot of serious bugs.

Few Examples that define the Need of Testing

- **Excel : 77.1 x 850 = 100,000 or 65,535**

  The bug in MS Excel is that when multiplying two numbers whose product equals 65,539 will actually give the answer as 100,000.
- **Excel incorrectly assumes that the year 1900 is a leap year**

  Excel is developed from Lotus 1-2-3. When Lotus 1-2-3 was first released, the program assumed that the year 1900 was a leap year, even though it actually was not a leap year. This made it easier

for the program to handle leap years and caused no harm to almost all date calculations in Lotus 1-2-3.

- **Y2K Problem in Payroll systems designed in1974**

In 1974, when first payroll system was developed, it was learnt that most of the memory space is utilised to store dates. And so, to minimize the space utility, the developers thought of storing the year of the date in 2 digits instead of 4 digits i.e. instead of storing 1900, store 00.

This idea of space conservation was OK then, but after 25 years, in the yr 2000, the question arose that how to store 2000 – if it is stored as 00 then would it represent 1900 or 2000?

So, all these above defects in the software are caused as the Software Testing was not followed from the initial phases of SDLC. Testing must not be performed in the last stage of SDLC just for the sake of doing it.

The cost of fixing a bug (error) and making the required changes in early phases of software development is less as compared to the same detected in later phases. This is because making changes in earlier phases is easy and if the bugs are detected in later phases, it becomes difficult to repeat the previous phases once again in order to make the required changes and thus, the cost goes high.

Testing must be done from initial SDLC phases to prevent and detect the defects so as to reduce cost and schedule risks.

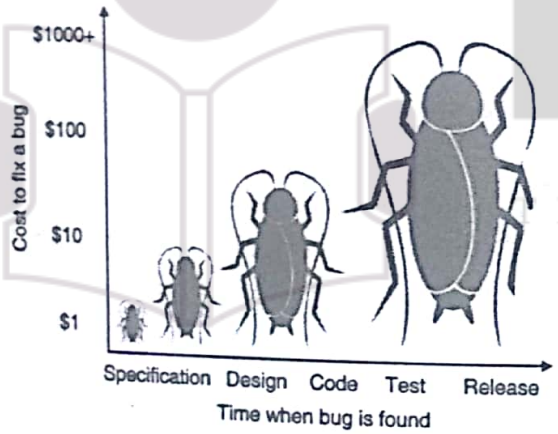


Fig. 11.2.1 : Cost of fixing bugs at different SDLC phases

Testing finds different kinds of non-conformances also called as errors. An error is a difference between the measured value and the specified value. Following are some types of errors :

1. **Syntax Error :** improper code.
2. **Logical Error :** mistake in the logic compiles and runs the program but produces incorrect output. *For example*; If we write a MACRO in C as SUM (-) by mistake and if we try to perform 'c = a SUM b' expecting to get the result of their addition but instead we get the output of subtraction as we have defined SUM to the (-).
3. **Run Time error :** This error is generated during the execution of the program such as array bound error, divide by zero error floating point error etc.

Goal of testing is to find ***bugs*** as early as possible and make sure they get fixed.

11.2.1 Bugs

A software bug occurs when :

1. The software doesn't do something that is defined in the SRS..

   Example : The SRS for a calculator states that it must perform four functions correctly i.e. addition, subtraction, multiplication, and division. Then, as a tester, if you press the ÷ key and nothing happens or if you a get a wrong answer then that's a bug according to this rule.

2. The software does something that is directly denied in the SRS.

   Example : The SRS for a calculator states that the calculator should never crash, lock up, or freeze. Then, as a tester, if you pound on the keys and the calculator stops responding to your input then that's a bug according to this rule.

3. The software does something that the SRS doesn't mention.

   Example : As a tester, if you find that besides addition, subtraction, multiplication, and division, the calculator also performs square roots since an ambitious programmer just included this function in to it because he felt it would be a great feature in the calculator. Then that's a bug according to this rule.

4. The software doesn't do something that the SRS doesn't mention but should do.

   The purpose of this rule is to catch things that were forgotten in the specification.

   Example : As a tester, if you start testing the calculator and come to know that when the battery gets weak, you no longer get correct answers for your calculations. It was assumed that the battery will be always fully charged. That means, correct calculations were not performed with weak batteries and it wasn't specified in the SRS about how it should react in such a situation but the calculator must either work properly until the batteries were completely dead or at least notify the user in some way about the weak battery. Then that's a bug according to this rule.

5. The software is difficult to understand, difficult to use or slow.

   Example : As a tester, if you find that the buttons of the calculator were too small or may be the placement of = key made it hard to use or may be the display is hard to read under bright lights, then that's a bug according to this rule.

Causes of Bugs

1. Incomplete or Erroneous Specification (IES): The root cause of software bugs is the incorrect requirement specification or constantly changing requirements.
2. Misinterpretation Of Customer Communication (MCC): Requirements are not communicated well to the entire development team.
3. The next main cause of bugs is the *design*.

   The first – second and third causes can be well described by an old saying, "If you can't say it, you can't do it."
4. Intentional deviation from SRS
5. Violation of programming standards
7. Inconsistent module interface
8. Error in designs
9. Incomplete testing
10. Inaccurate or incomplete documentation
11. Difficult human-computer interface

Bug Life Cycle

Q. Define the terms : Bug life cycle

- Bugs can occur at any stage of SDLC process. The bug then attains different states which can be termed as its life cycle.
- The bug life cycle begins when the developer makes mistakes and the cycle ends when the bug is fixed and it is no longer in existence.
- The bug life cycle is shown diagrammatically as follows :

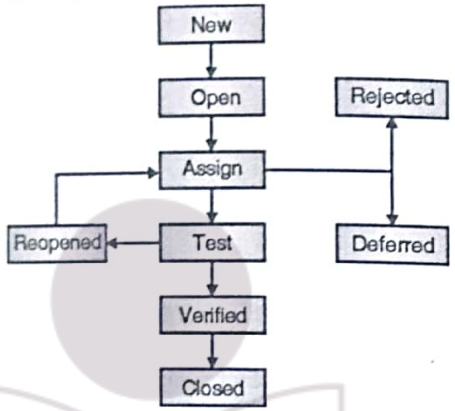


Fig. 11.2.2 : Bug Life Cycle

The different states of a bug in the can be described as follows :

1. **New :** When the bug is detected for the first time, its state is called as NEW. This means that the bug is not yet approved as a bug.
2. **Open :** After detecting the bug, the leader of the tester approves that the bug is genuine needs to be fixed; thus, he changes its state as OPEN.
3. **Assign :** After changing its state as OPEN, the test leader assigns the bug to corresponding developer or developer team. The state of the bug is now as ASSIGN.
4. **Test :** The developer then fixes the bug and re-assigns the bug to the testing team for next round of testing. The state of the bug is now as TEST. It specifies that the bug has been fixed and is released to testing team for re-testing.
5. **Deferred :** The bug changed to DEFERRED state means that the bug is expected to be fixed in next releases. The reasons for changing the bug to this state might be that the priority of the bug may be low, or lack of time in release of the software or the bug may not have major affect on the software.
6. **Rejected :** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is now as REJECTED.
7. **Duplicate :** If the same bug repeats twice then that bug status is changed to DUPLICATE.
8. **Verified :** Once the bug is fixed, the tester tests the bug. If the bug is no more present, the tester changes its state to VERIFIED.
9. **Reopened :** If the bug still exists in the software even after the bug is fixed by the developer, then the tester changes its state to REOPENED. The bug traverses through the bug life cycle once again by assigning it to the developer.

10. **Closed :** Once the bug is fixed, it is tested by the tester and if he feels that the bug no longer exists in the software and it will not even crop up in the future, then he changes the bug state to CLOSED which means that the bug is fixed, tested and approved.

The tester or the developer decides the severity of the bug on the basis of the priorities assigned to the bugs. *There are different types of priorities assigned to the bugs.*

1. **Critical / Show's Stopper :** A bug that prevents further testing of the product is called as Critical bug.

   Examples : A missing menu option to access a function under test.
2. **Major / High :** A bug that does not function as expected or cause other functionalities to fail to meet the specified requirements is called as Major Bug.

   Examples: inaccurate calculations.
3. **Average / Medium :** A bug that does not conform to standards and conventions is called as Medium Bug.

   Examples : Visual and text links that may lead to different end points. .
4. **Minor / Low :** look and feel bugs that do not affect the functionality of the system is called as Minor Bug.

Syllabus Topic : Testing Principles

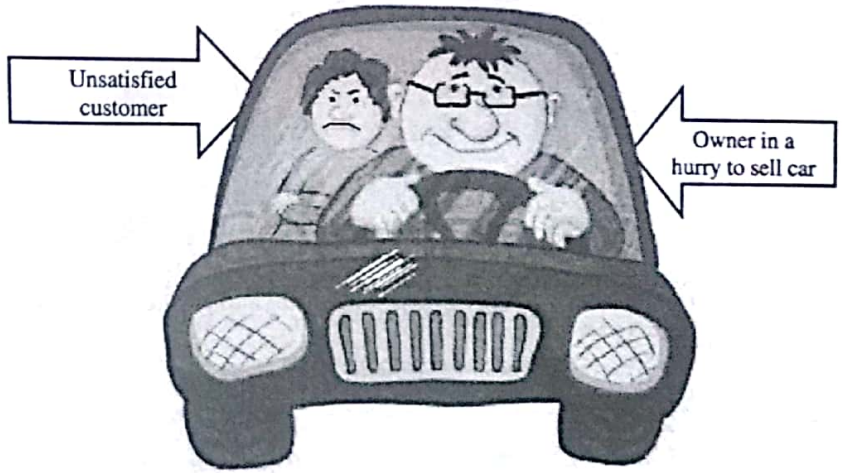
11.3 Testing Principles

Below is a list of ***testing principles*** that serves as a basis for testing objectives so as to provide quality products to customers.

1. Testing should focus on finding defects before customers find them.

   The software product should be thoroughly tested to ensure that the software product is defect free before delivering it to the customer; else if the customer comes across the defects at the installation time, then he becomes unsatisfied and will not be ready to buy it.

   Example : An interesting act between car owner and car buyer (on phone)



Car owner : Hello Mr. Amol, you can come today and take the car. Customer Amol Arrives to car owner's house and sees the car that he is going to buy in a very bad condition…

Customer : What's this? Is this the car, you want to sell. Oh no! How dirty it looks. There are so many crashes and the color has also fainted.

Car owner : Chill…Pill… Mr. Amol. The car is complete… you just need to paint it.

Customer : Ok… First let me take a test drive. …Both sit in the Car…

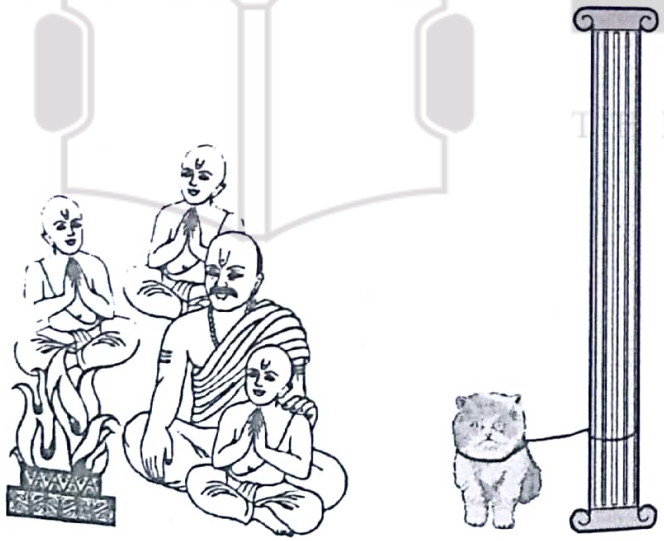
Car owner : Mr. John, this car has the best possible transmission and brake, and accelerates from 0 to 80 kmph in under 20 seconds!

Customer : Well. It may be true. But unfortunately it accelerates even faster when I press the brakes.

Moral : The car owner was trying to sell his second hand car without repairing the defects in it. When customer came to know about these defects, he refused to buy the car.

2. Testing is applied all through the SDLC process and not just at the end of the SDLC activity.
Example : This principle can be better understood from the graph describing the 'Cost of fixing bugs at various SDLC phases'.

3. Understand the reason behind test - "what to test" and "how to test"
Before performing any test, you should understand what product you will be testing, what is the expected outcome of that product, why you are performing any test. Else you will end up with inappropriate tests that will not address what the product should do.
Example : An interesting act in an Ashram between Saint and his Disciples



Saint : Ok. Disciples, I will sit here and meditate for a while. So, please, see to it that there is no disturbance.

Disciple : Ok. Sir. …A cat comes there and disturbs the Saint. So, disciples tie the cat to a pillar…..

Disciples : Ooops! This cat is creating a lot of nuisance. Let's tie it to a pillar. After five years… …Disciples are running here and there, searching for a cat…

Disciple : Oh shit! I'm unable to find any cat. What to do. A guest in the Ashram asks the disciple: Sir, what are you searching for?

Disciple : We need a cat. Only when we get a cat, then we can tie it to a pillar and only after that, our saint will start meditating.

Moral : The daily routine of catching a cat and tying it to a pillar becomes a tradition. And the further generations, without any actually understanding that why the cat was tied; they everyday searched for a cat by hook or crook and tied it to a pillar even if it didn't come there to disturb. It is just ridiculous to catch a cat without understanding why it is caught.

4. Test the Tests (testing tools) First - Defective testing tools are more dangerous than a defective product. Testing tool is also ultimately software produced by human beings. So, we cannot assure that testing tools are perfect. So, ensure that testing tools are not faulty before you use them.
Example : An interesting act in between ENT (Ear Nose Tongue) doctor and his patient

Doctor : I want to test the range within which you can hear. I will ask you from various distances to tell me your name, and you should tell me your name.

Patient : Ok. Sir. I understood. ..(doctor 30 feet far)…

Doctor : what is your name?

Doctor : I cannot listen to her answer. I think she cannot hear me. …doctor goes at 20 feet…

Doctor : what is your name?

Doctor : I cannot listen to her answer. I think she still cannot hear me. …from 10 feet…

Doctor : What is your name?

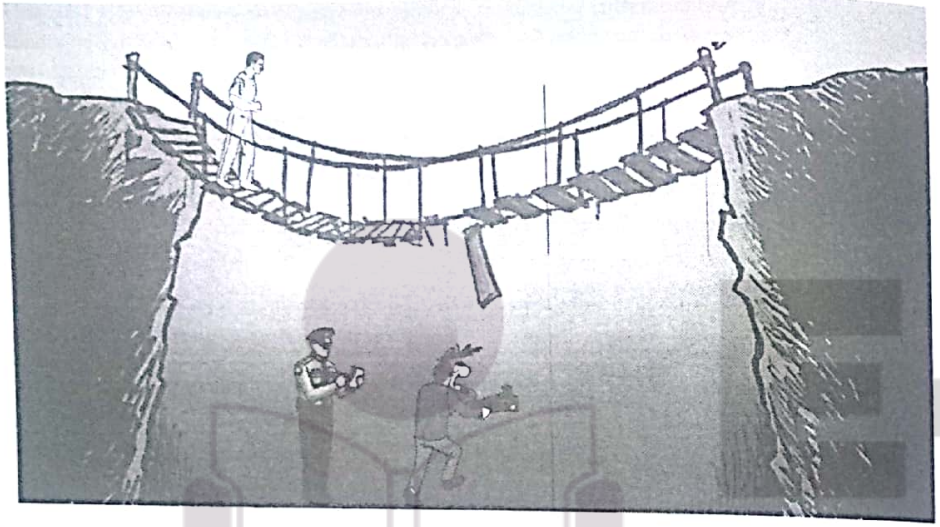
Patient : For the third time, I am repeating my answer, my name is Richa.

Moral : The ENT doctor himself had a defect in his ear. So, how can he succeed in testing the patient i.e. a faulty and a defective doctor cannot test a patient.

5. Tests develop immunity and have to be revised constantly.
Example : Tests are like pesticides that need to be constantly revised to tackle the new pests i.e. defects

6. Testing encompasses on defect prevention because Prevention is better than cure
Example : If you know that you are having low haemoglobin, then start eating 'Gu-chan i.e. jaggery and groundnut' to avoid fainting on some day.
Example : Assume an over bridge and people are passing over it. And what happens is that whenever a person passes over the bridge, he falls down the bridge. No one knew what's the reason? Then Government appointed a police under the bridge to catch the persons who are falling

down the bridge. But what if the falling person is too fat and police is too thin. And this is not the proper solution. It's better to find the reason behind the falling so that we can prevent the accident in futures. This will not waste the police officers time spent in standing over there and also he can spend his mind and knowledge in some other work at the same time. There we say that "prevention of falling down of persons is better than waiting for the persons to fall then curing them"



7. Defect prevention and defect detection should supplement each other and not be considered as mutually exclusive.
8. Defect Prevention improves the quality of the process while defect detection and testing is needed to catch and correct the defects that escape the process. Both are necessary for producing a quality product.

Example : An interesting act between three Chinese doctor brothers. ...One is a Surgeon... famous all over the world...

Surgeon to the patient	:	Hmm... You are suffering from tumor. No problem. I will do an operation and remove this tumor. All will be well soon. ...One is a doctor... famous in the city they lived...
Doctor to the patient	:	hmm... You are suffering from Jaundice. No problem. This is the first stage. Take these tablets. You will get cured very soon. ...But, Both Surgeon and Doctor always took advice of their youngest brother.
Doctor to his youngest	:	Tell me how to prevent from Cholesterol. Brother
Younger Brother	:	Avoid oily food; take 'Oats' in the breakfast and exercise daily and have balanced diet.
Doctor	:	Wow! Thanks for your preventive measure.
Surgeon to his youngest	:	Brother. Tell me how to prevent from Blood brother Cancer.

Younger Brother	:	Avoid Smoking, avoid taking drinks and avoid intake of Nicotine through Gutka and as such.
Surgeon	:	Oh. Thank you. You know preventive measures for all the defects. Your advice is very effective.
Moral	:	Preventing an illness is more effective than curing it. People who prevent defects usually do not get much attention. They are usually unseen heroes of an organization and those who put out the fires are the ones who get visibility. This however, should not demotivate the people who do the defect prevention.

Syllabus Topic : Testing Objectives

11.4 Testing Objectives

1. To execute a program with the intent of finding an *error*.
2. To verify and validate if the system meets the requirements and be executed successfully in the Intended environment.
3. To verify and validate if the system is "Fit for purpose".
4. To verify and validate if the system does what it is expected to do.
5. Exhaustive testing is not possible that means the testing can only trace the presence of defects and never their absence.
6. Testing should begin 'in the small' and progress towards testing 'in the large'.

   Example : First, the unit testing is performed to check each and every code line; then black-box testing is performed to test the functions; then integration testing to test the individual components and the integration of components; at last the system testing is done to test the overall working of the system.
7. Intelligent and well planned automation is the key to realizing the benefits of testing. That means, tests should be planned long before the testing begins
   - First know the reason behind why you want to automate the tests
   - Analyze the various tools before choosing on s being most appropriate for your need

- Choose the tools that match your needs rather than changing your needs to match the tool capabilities.

Syllabus Topic : Test Oracles

11.5 Test Oracles

A function that determines if the application has behaved correctly in response to a test action is called a *test oracle.*

The test oracle is either of the following

- a program (separate from the system under test) which takes the same input and produces the same output
- documentation that gives specific correct outputs for specific given inputs
- a documented algorithm that a human could use to calculate correct outputs for given inputs
- a human domain expert who can somehow look at the output and tell whether it is correct
- or any other way of telling that output is correct.

Test oracle is a mechanism for determining whether a test has passed or failed.

A **test oracle** may be the existing system (for a benchmark), other software, a user manual, or an individual's specialized knowledge.

☞ **Some issues faced while testing a large and complex applications**

- It may require millions of test runs.
- Size of the outputs might exceed the expectation

So, automated oracles are essential to overcome these issues.

The use of test oracles involves comparing the actual output(s) of the system under test for a given test-case input with the expected output(s) that the oracle determines that product should have.

A good test oracle would have three capabilities :

1. A *generator* - provides expected results for each test.
2. A *comparator* - compares expected and actual results.
3. An *evaluator* - determines whether the test passed or failed

Test Oracles can be built from :

- **System specification :** System oracles are designed early to check the specified properties and Subsystem oracles are a part of architectural design and system build plan.
- Designs

Example 1 : A UML sequence diagram indicates a test case and expected outcome. A scenario based oracle can be used.

Example 2 : A UML state diagram indicates all permissible behaviors of a module. A oracle can be used with large numbers of automatically generated test cases.

- Code documentation

Economics of Test Oracles :

- Expected output for each test input
  - Easy to manually verify for one test input
  - Expensive to verify for many test inputs

- Properties applicable for multiple test inputs
  - Not easy to write

Syllabus Topic : Levels of Testing

11.6 Levels of Testing

V&V model illustrates different **levels of tests** at different **phases of SDLC.**

SDLC phases of the *V-model are as follows :*

- Business Requirements (BRS) and System Requirements (SRS) begin the life cycle. The test plan is created that focuses on meeting the functionality specified in the SRS.
- High-level Design (HLD) phase focuses on system architecture and design. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.
- Low-Level Design (LLD) phase is where the actual logic for each and every component of the system is designed. Class diagram with all the methods and relation between classes comes under LLD.
- Code (Implementation) phase is where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Various levels of testing are explained in below sub sections.

11.6.1 Level 1 : Unit Testing

- It is the most micro scale of testing where each and every unit of the system is tested.
- Unit testing tests the individual Chunks composed of objects, functions, procedures or subroutines, or any other small units of a Software Product.
- The unit testing is performed parallel to development of each unit.
- The idea behind unit testing is to test each small unit or say, every building block of a program before combining them to build a whole system.
- Unit testing is usually done by the developers if they are ought to dynamically test the code and it is done by the inspection team or review team if it is ought to just verify the code documents.
- Unit testing requires technical knowledge of the internal design and programming language used and the logic also needs to be strong.
  - *Objectives* :To test the function of a smallest executable unit
  - To conduct statement coverage, conditional coverage and path coverage.
  - To test exception & error handling

Done by : Programmers
Debuggers
Tracers

Done : After modules have been coded

Input : Internal application descriptions
Unit test plan

Output : Unit test report

Methods : Static testing or White box (structural) testing techniques.

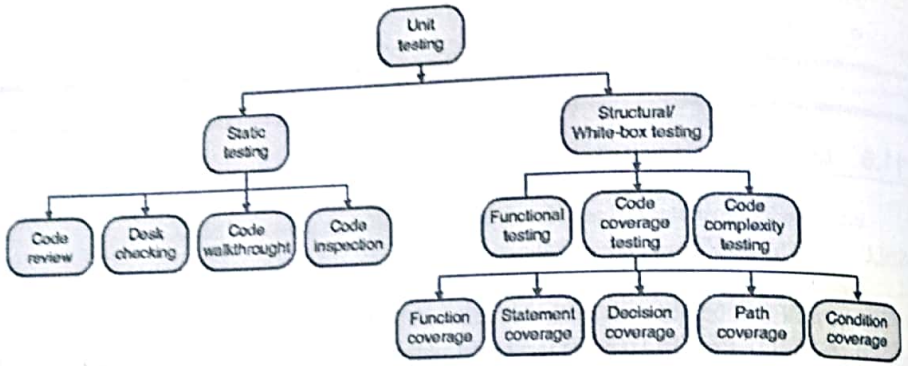


Fig. 11.6.1 : Types of Unit Testing

☞ **Advantages**

1. Unit testing covers logical analysis of software elements.
2. Unit testing allows multiple units to be tested simultaneously and also, the testing activities can proceed in parallel with development activities.
3. Unit testing is focused on testing the programming code, the finding and correcting of defects is easier.
4. Thus, unit testing helps to build an entire system with defect free units that ultimately achieve higher quality software in less time.

11.6.2 Level 2 : Component Testing

- Component testing finds the defects in the module and verifies the functioning of software.
- Component testing may be done independent from rest of the system testing. In such a case the missing software is replaced by Stubs and Drivers and simulate the interface between the software components in a simple manner.
  - o A stub is called from the software component to be tested.
  - o A driver calls the component to be tested.

Example : Suppose there is an application consisting of three modules say, module A, B, C. The developer has developed the module B and now wanted to test it. But functionality of module B depends on module A and module C. But module A and module C is not yet developed. In this case to test the module B, we should replace module A and module C by stub and drivers as required.

11.6.3 Level 3 : Integration Testing

- Integration testing is also called as Component testing because it is about testing whether the system works properly when different components are integrated to form a complete system.
- Integration is defined as set of interactions among components. Testing the interaction between modules and interaction with other systems externally is called integration testing.

- Integration testing goes through internal and external interfaces, and tests the functionality of the software.

Objectives : To verify the interfacing between modules, and within sub-systems.
Done by : Developers
Testers
Done : After modules are unit tested and also all the functions of each component are tested.
Input : Internal and external application descriptions
Integration test plan
Output : Integration test report
Methods : Top-down and bottom-up testing techniques.

- Integration testing can be classified into four broad categories depending upon the order in which the subsystems (components) are selected for testing and integration.
  1. Bottom up integration testing.
  2. Top down integration testing.
  3. Big bang integration (Non-incremental) testing.
  4. Sandwich testing.

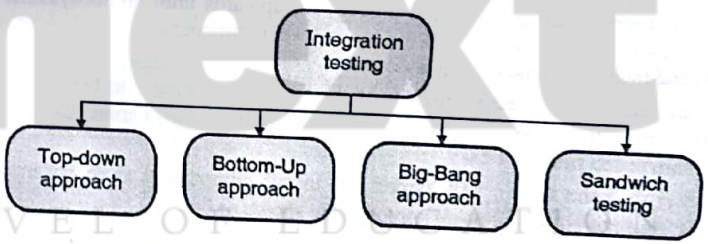


Fig. 11.6.2 : Types of Integration Testing

11.6.3(A) Top-down Testing

- This is about breaking down of a system in to its sub components so as to study its compositional sub-systems in greater detail. Consider the Fig. 11.6.3 which depicts the breaking down of a system into its sub-systems (components).

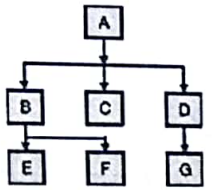


Fig. 11.6.3 : System 'A' divided into its sub components

- In top-down approach, the testers first test the top layer of the system and proceed downwards until all subsystems are incorporated into the test - Components which are at the top layer are tested first and then it is integrated with the just below components and then tested it and so on.

Example : Consider the system 'A' divided into its sub components as shown in Fig. 11.6.3. Therefore, the testing steps with top-down approach are as shown as below;

Fig. 11.6.4 : Top-down Testing

- The importance of the top – down testing is design flow is detected, quality improves, user satisfaction, requirements matches with the developed software, user view primary view of the system, then secondary view of the system and so on.

☞ **Advantages**

- Detects the flow of design.
- Useful when major flaws occur at the top of the program.

11.6.3(B) Bottom-up Testing

- In this approach, the sub systems are linked together to form larger subsystems which then in turn are linked sometimes in many levels, until a complete top-level system is formed.
- In the bottom-up approach the individual base elements i.e. the elements at the lower level of the system are first tested in great detail and proceeding upwards until all subsystems are combined together to forma a whole system i.e. all the lower level components are integrated with just above components and then combined testing is done.
- Bottom-up approach is completely opposite of the top-down testing – in bottom-up approach, the components at the bottom level are tested first; where as in top-down approach, the components at the top level are tested first.

  Example : Consider the system 'A' divided into its sub components as shown in the Fig. 11.6.3. Then, the testing steps with bottom-up approach are shown as below;

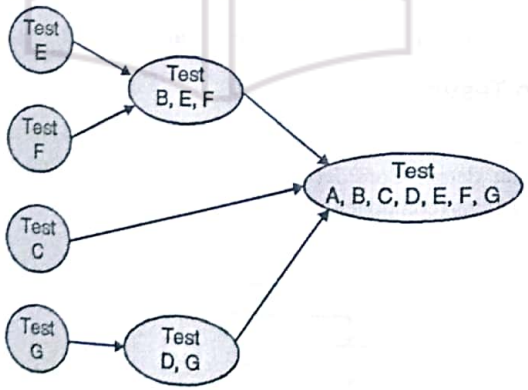


Fig. 11.6.5 : Bottom-up Testing

☞ **Test Driver**

- The test drivers are used during Bottom-up integration testing in order to simulate the behavior of the upper level modules that are not yet developed or integrated.
- Test drivers are the modules that act as temporary replacement for a calling module.

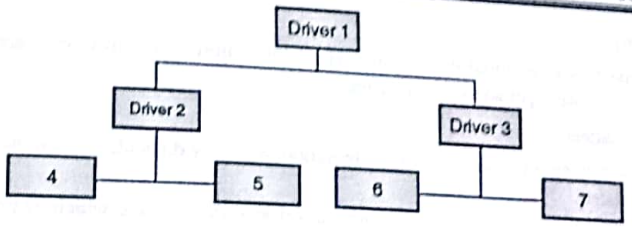


Fig. 11.6.6

- From the Fig. 11.6.6 , it is clear that the modules 4,5,6,7 cannot be integrated until the above modules are integrated. And the above modules are still under development that cannot be integrated at this point of time.

☞ **Advantages**

- Useful when major flaws occur at the bottom of the program.
- Useful for integrating the Object-oriented systems
- Test conditions are easier to create.
- Observation of test results is easier.

☞ **Drawbacks**

- Integration errors are found later than earlier.
- Design flaws that could need major reconstruction are found last.

11.6.3(C) Big-Bang Approach

- This approach is ideal for a product where the interfaces are stable with less number of defects.
- In big – bang testing, all smallest components in the product are tested individually then all of them are combined together and tested as a combinational work of the system.
- This approach is used to test how one component supports the other component.

  Example : Consider the system 'A' divided into its sub components as shown in the Fig. 11.6.3. Then, the testing steps with big-bang approach are shown as in Fig. 11.6.7;

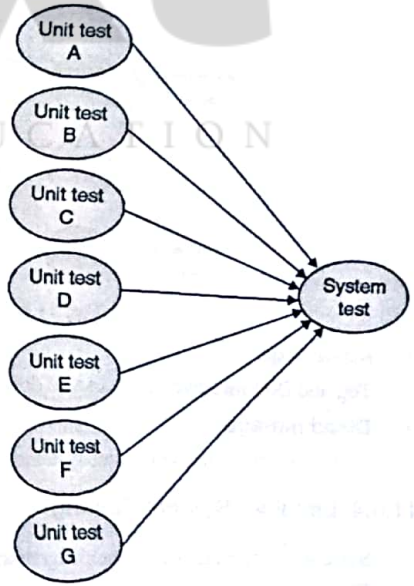


Fig. 11.6.7 : B

Advantage

This approach is well suited in scenarios where the majority of components are already available and very few components get added or modified.

Disadvantages

- When some failure is found out during integration, it is very difficult to trace the module where the problem existed.
- It is also difficult to find the developer who developed that module which again makes it difficult to make modifications so as to correct that defect.
- Certain sub-systems may take lot of time to be corrected.

11.6.3(D) Sandwich Testing

- This is the combination of top -down and bottom-up testing.
- This approach *views the system as if having three layers* and the testing converges at the target layer
  1. Target layer in the middle
  2. layer above the target
  3. layer below the target.

Example : Consider the system 'A' divided into its sub components as shown in the Fig. 11.6.3 Then, the testing with sandwich approach is shown as in Fig. 11.6.8;

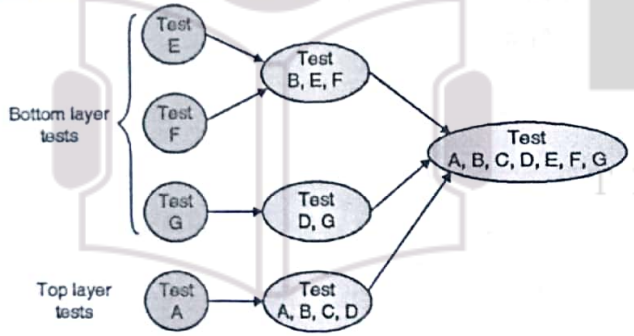


Fig. 11.6.8 : Sandwich Approach

Advantage

Top and Bottom Layer Tests can be done in parallel.

Disadvantage

Does not test the individual subsystems thoroughly before integration of the sub components.

11.6.4 Level 4 : System Testing

- System testing begins after the integration of all tested components is completed.
- The main objective of system testing is to determine if the integrated components work together as designed.
- A system is usually defined as a set of hardware, software and other parts that integrate to provide product features and solutions.

- System testing is performed on completely integrated components and solutions to estimate the system compliance with specified requirements (the requirements may be either functional or non-functional).
- This testing brings out issues that are fundamental to design, architecture and code of the whole product.

Objectives : To determine that system as a whole fulfills all functional and non-functional requirements.
Done by : Developers
Testers
Users
Done : After integration testing
Input : Detailed requirements and external application descriptions
System test plan
Output : System test report
Methods : Performance, load, stress, security testing techniques

- All the *system testing types* are ***Non-Functional Testing*** types since it is not about testing the functionality of the program code; instead, it is about testing the behaviour and security issues of the software.

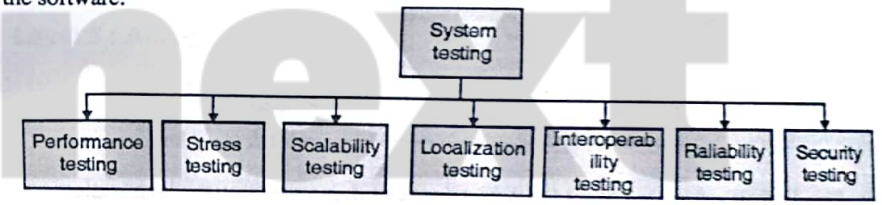


Fig. 11.6.9 : Types of System Testing

1. Performance testing

- Each and every system has implicit performance requirements such as - the software shouldn't take endless time or endless resources to execute. 'Performance bugs' are used to represent the design problems in the software may degrade the system's performance.
- Performance evaluation of a software includes resource usage, throughput, stimulus-response time and queue lengths describing the average or maximum number of tasks waiting to be serviced by selected resources (resource may be the network bandwidth, CPU cycles, memory or disk space and disk access operations).
- The goal of performance testing is to ensure that a product :
  - Number of transactions processed in any given time interval – ***Throughput.***
  - Availability of system under different load conditions – ***Availability.***
  - Time taken between request and response of a page - ***Response Time.***
  - Optimum number of resources used to accomplish a request i.e. the percentage of time a component (CPU, Channel, storage, file server) is busy - ***Capacity Planning.***

2. Stress testing

- This is done to find out whether the product's behavior degrades under extreme conditions such as maximum number of users, peak demands or extended number of operations or transactions.

- The product is over-stressed deliberately to simulate the resource crunch and to find out its behavior. It is expected to gracefully degrade on increasing the stress but the system is not expected to crash at any point of time during stress testing.
- It helps in understanding how the system behaves under extreme and realistic situations.

3. Scalability/Volume/Load testing

- This is done to find out whether the product's behavior degrades under extreme conditions such as what happens if large amounts of data are handled at a time on the same hardware and software configurations.
- This testing requires enormous amount of resource to find out the maximum capability of the system parameters.

4. Localization testing

- It is about global functioning of the product. It will verify that the application still works, even after it has been translated into a new language or adapted for a new culture.

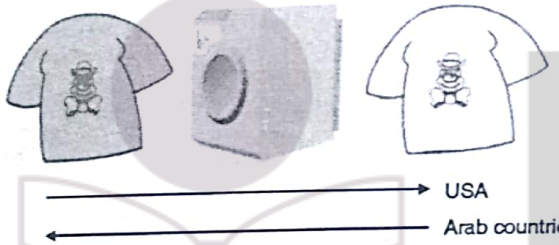


Fig. 11.6.10 : Washing Machine Advertisement demonstrating the localization concept when read from right to left and vice versa

- In the Fig. 11.6.10, you will see that the meaning of this advertisement when read from left to right is 'dirty cloth kept in washing machine gives clean cloth' but the meaning changes when it is read from right to left as is the trend of reading in Arab countries. In this case, the advertisement says, 'clean cloth kept in washing machine gives dirty cloth'. Therefore, make sure that the software works in the same manner even after it has been translated into a new language or adapted for a new culture.
- It ensures the software product should work in the same manner even if its User Interface language is transformed into any other language.

  Example : Google services the client in the same manner, no matter client opens it in English or Hindi or any other language.

5. Interoperability testing

It ensures that two or more products can exchange information, use the information, and work closely.

6. Reliability testing

To verify the ability of the sub-system or a system to confirm that it performs the user required functions without any frequent errors. It is to check that how long and how efficiently system works without any error.

7. Security testing

- Security Testing verifies that protection mechanisms built into the system will protect it from improper penetration.
- Security testing is the process of executing test cases that subvert the program's security checks.

  Example :
  - One tries to break the operating systems memory protection mechanisms
  - One tries to subvert the DBMS's data security mechanisms
- The role of the developer is to make penetration cost more than the value of the information that will be obtained.

8. Recovery testing

- It verifies whether the system recovers completely from expected or unexpected events without loss of data or functionality.
- The events that may cause data loss may be such as shortage of disk space, unexpected loss of communication or power out conditions.
- It tests how well a system recovers from crashes, hardware failures or other problems.

11.6.5 Level 5 : Acceptance Testing

- This testing is performed by end-users or representatives of the end-users. The customer defines a set of test cases that are executed to check whether the product meets the user requirements and only then, it is qualified and accepted as the product. These test cases are executed by the customers themselves in order to judge the product before deciding to buy it.
- Test cases for Acceptance testing are executed to verify if the product meets the acceptance criteria defined during the software requirement specification phase of the SDLC.

Objectives	:	To determine that the system meets the user requirements.
Done by	:	Users
Done	:	After System Testing
Input	:	Business Needs & Detailed Requirements User Acceptance Test Plan
Output	:	User Acceptance test report
Methods	:	Normally performs black-box techniques to check the system functions.

- Acceptance testing can be classified into two broad categories depending upon the place where it can be conducted;

1. Alpha testing 2. Beta testing

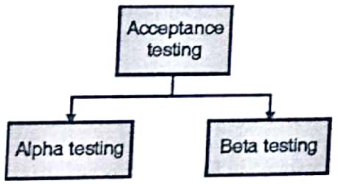


Fig. 11.6.11 : Types of Acceptance Testing

1. **Alpha testing**
   - Alpha testing is done when development is nearing to completion and some minor design changes may still be made in the system as a result of alpha testing.
   - This is conducted at the developer's site by end users. The software is used in a natural setting with the developer "looking over its shoulder" of typical users and recording errors and usage problems. Alpha tests are conducted in a controlled environment.
   - Sponsorer/customer uses the software at the developer's site.
   - Software is used in a controlled setting, with the developer always ready to fix bugs.
2. **Beta testing**
   - This is conducted at end-user sites. Unlike alpha testing, the developer is generally not present in beta testing. Therefore, beta testing is also called as 'live' testing of software as it is done in an environment where it is not controlled by any developer.
   - The end-user documents all errors that occur during beta testing and submits the report to the developer at regular intervals. As a result of this error report generated through beta testing, software engineers make modifications and then start preparing for the release of the software product.
   - Conducted at *sponsor's site* (developer is not present).
   - Software gets a realistic workout in target environ-ment.
   - Potential customer might get discouraged.

11.6.6 Integration Testing vs. System Testing

Q. Differentiate: Integration and System Testing

Table 11.6.1 : Difference Between System Testing And Integration Testing

Sr. No.	Integration Testing	System Testing
1.	When integration testing starts once the functional testing on individual components is finished.	When System testing begins after the integration of all tested components is completed.
2.	Objective : To verify the interfacing between modules, and within sub-systems	Objective : To determine if the integrated components work together as designed.
3.	Aim : Breaks down the system into sub systems or links the sub systems into a system to ensure that the interfacing of these subsystems/components/modules doesn't affect the functioning of each other and thus work properly when integrated with each other.	Aim : This testing brings out issues that are fundamental to design, architecture and code of the whole product.
4.	Methods : Top-down integration, bottom-up integration testing, big bang integration testing and sandwich testing.	Methods : Performance, load, stress, security testing techniques
5.	Integration testing is also called as Component testing because it is about testing whether the system works properly when different components are integrated to form a complete system.	All the system testing types are Non-Functional Testing types since it is not about testing the functionality of the program code; instead, it is about testing the behaviour and security issues of the software

11.7 Static Verification

- It is a ***verification*** process
- Testing a software without execution on a computer. Involves just examination/review and evaluation
- It is done to test that software confirms to its SRS i.e. user specified requirements
- It is done for ***preventing*** the defects
- Static Testing is a process of reviewing the work product and reviewing is done using a checklist
- Static Testing helps weed out many errors/bugs at an early stage
- Static Testing lays strict emphasis on conforming to specifications
- Static Testing can discover dead codes, infinite loops, uninitialized and unused variables, standard violations and is effective in finding 30-70% of errors.

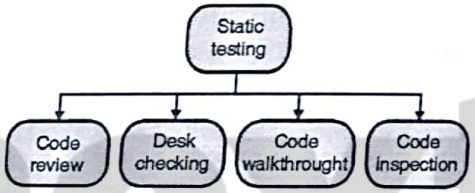


Fig. 11.7.1

11.7.1 Code Review

Q. Define the terms :Code Review

- Code reviews are extremely cost-effective strategies to reduce coding errors and produce high quality code.

Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned? Is there any possibility of buffer overflow?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for? If a break is required after each case in case statements, has it been included?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output? Can unexpected inputs cause corruption?

- Normally two types of code reviews are carried out on the code of the module: *code walk-through* and *code inspection.*

11.7.2 Desk Checking

Q. Define the terms : Desk-checking

- This is done by authors of the code. It is a method to verify the portions of the code for correctness. Such verification is done by comparing the code with the design or specification to make sure that the code does what it is supposed to do and effectively. This method of catching and correcting errors is characterized by :
- No structured method or formalism to ensure completeness
- No maintaining of a log or checklist.
- This method completely depends on authors'/developers' thoroughness, attentiveness and skills.

Advantages

- The programmer who is good in programming language is able to read and understand his system's code.
- As the desk checking is done by an individual, there are fewer scheduling and logistic expenses.

Disadvantages

- A developer is not the right person to detect problems in his or her own code.
- Developers generally prefer to write new code rather than do any form of testing.
- This method is essentially person-dependent and informal and thus may not work consistently across all developers.

11.7.3 Code Walk-Through

Q. Define the terms : Walk-through

- Code walk-through is an informal code of analysis technique. In this technique after a module has been coded, it is successfully compiled and all syntax errors are eliminated.
- In walkthroughs, a set of people look at the program code and answer the questions. If the author is unable to answer some questions, he or she then takes those questions and finds their answers.
- Some members of the development team are given the code a few days before the walkthrough meeting to read and understand the code.
- Each member selects some test cases and simulates execution of the code by hand.
- The main objectives of walk-through are to discover the algorithmic and logical errors in the code.

Guidelines to perform code walkthrough

- The team performing the code walk-through should not be either too big or too small. Ideally, it should consist of three to seven members.
- Discussion should focus on discovery of errors and not on how to fix the discovered errors.
- In order to promote cooperation and to avoid the feeling among the engineers that they are being evaluated in the code walk through meeting, managers should not attend the walk through meetings.

Advantage over desk checking

Walkthroughs bring multiple perspectives.

Disadvantage

Completeness is limited to the area where questions are raised by the team.

11.7.4 Code Inspection

Q. Define the terms : Inspection

- The inspection takes place only when the author has made sure the code is ready for inspection by performing some basic desk checking and walkthroughs.
- *List of some programming errors which can be checked during code inspection* :
  - uninitialized variables
  - Non terminating loops
  - Array indices out of bounds
  - Improper storage allocation and deallocation
  - Reading a file that doesn't exist
  - Arithmetic exception i.e. Divide by zero
  - Mismatch between function definition and function call due to mismatch between actual and formal parameters passed.

Inspections and Testing

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V and V process.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

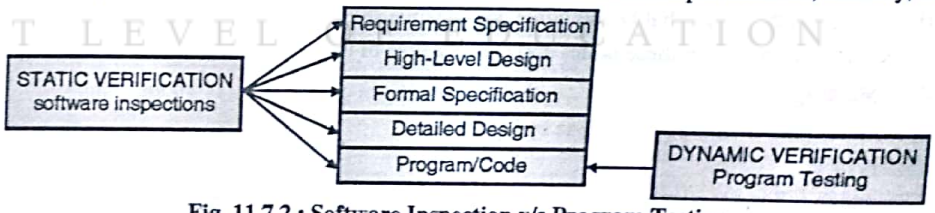


Fig. 11.7.2 : Software Inspection v/s Program Testing

Inspection pre-conditions

- A precise specification must be available.
- Team members must be familiar with the organisation standards.
- Syntactically correct code or other system representations must be available.
- An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process.
- Management should not use inspections for staff appraisal i.e. finding out who makes mistakes.

Inspection Process

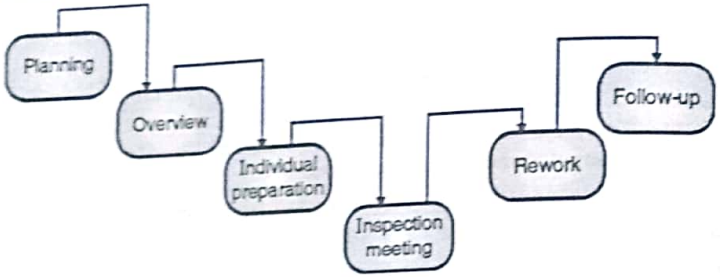


Fig. 11.7.3 : Inspection process

Step 1 : Planning

- Select the group review team - three to five people group is best
- Identify the moderator - has the main responsibility for the inspection
- Prepare work product for review plus supporting docs

Step 2 : Overview

- A brief meeting - explain purpose of the review, intro.
- All team members then individually review the work product
- Lists the issues/problems they find in the self-preparation log
- Checklists, guidelines are used

Step 3 : Preparation

- Each reviewer studies the project individually.
- He notes down the issues that he has come across while studying the project.
- He decides how to put up these issues and makes a note of it.

Step 4 : Meeting

- A reviewer goes over the product line by line
- Others raise the issues
- Discussion follows to identify if a defect
- Scribe records the issues
- If few defects, the work product is accepted; else it might be asked for another review
- Group does not propose solutions - though some suggestions may be recorded

Step 5 and 6 : Rework and Follow Up

- Defects in the defects list are fixed later by the Author. These modifications are made to repair the discovered errors.
- Once fixed, author gets it OKed by the moderator, or goes for another review
- Re-inspection may or may not be required.
- Once all defects/issues are satisfactorily addressed and reviewed, the collected data is submitted.

Inspection Roles

There are four roles in inspection: ***author*** of the code, ***moderator*** who is expected to formally run the inspection according to the process, ***inspectors*** who actually provide review comments for the code. Finally, there is a ***scribe*** who takes detailed notes during the inspection meeting and circulates them to the inspection team after the meeting.

Author or Owner	The programmer who develops and fixes the code
Inspector	Finds errors, omissions and inconsistencies in programs and documents.
Reader	Presents the code an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or Moderator	facilitates the inspection meeting.
Chief Moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Advantages

The goal of this method is to detect all faults, violations, and other side-effects. This method finds out more and more number of defects by:

- Demanding thorough preparation before an inspection.
- Enlisting multiple diverse views.
- Assigning specific roles to the multiple participants
- Going sequentially through the code in a structured manner.

Disadvantages

- Time consuming as it needs preparation as well as formal meetings.
- Logistics and Scheduling can become an issue since multiple people are involved.
- It is not always possible to go through every line of code with several parameters and their combinations to ensure the correctness of the logic, side side-effects and appropriate error handling.

Syllabus Topic : White-box Testing / Structural Testing

11.8 White-box Testing (Structural Testing)

- It is also called as 'Glass Box testing' because all the internal coding is clearly seen and tested. The internal structure of the system is as transparent and as clear as glass box and that's why it is called so. This is the same reason why it is also called as white-box testing.

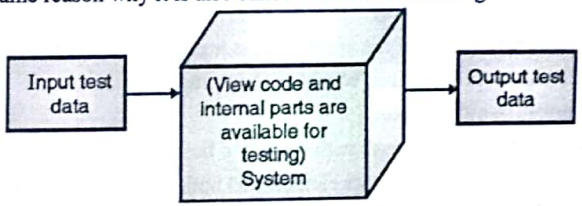


Fig. 11.8.1 : White Box Testing

- White Box Testing involves the complete knowledge of internal structure of source code and it can be used to find the number of test cases required to guarantee a given level of test coverage.
- In white-box testing (sometimes called clear-box testing), the software tester has access to the program's code and can examine it for clues to help him with his testing- he can see inside the box.
- It is error based testing. This kind of testing searches the given class's method for particular choice of interest and then describes how these choices to be tested.
- White Box test covers following contents;
  - Logical Analysis of software elements.
  - Possible paths of control flow.
- Software is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester.

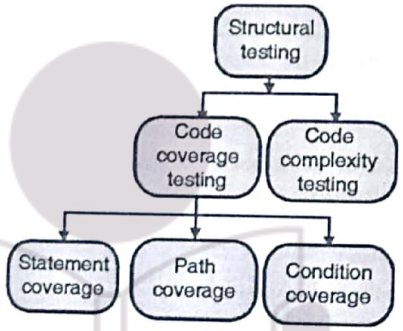


Fig. 11.8.2: Types of White box Testing

- White-box testing deals with the code, code structure, and internal design and how they are coded.
- Structural testing is actually run on the built product while Static testing is performed by humans just on the source code and not the executables.
- White-box testing needs knowledge of the internal program design and code required and these type of tests are based on coverage of code statements, branches, paths and conditions.
- Therefore, the structural tests are logic driven.

☞ **Structural testing involves**

- Testing all independent paths at least once
- Testing all logical decisions on their *true* and *false* sides
- Testing all loops at their boundaries and within their operational bounds
- Testing internal data structures to ensure their validity

☞ **Advantages of white-box Testing**

- The internal structure (code) of the application is tested thoroughly to ensure the validity.
- Forces the test developer to reason carefully about implementation
- Reveals the errors in hidden code.
- Spots the dead code or other issues with respect to best programming practices.
- Does the logical analysis of software elements on both true and false sides
- Tests all possible paths of control flow.

11.8.1 Code Coverage Testing

- This involves designing and executing test cases and finding out the percentage of code that is covered by testing.
- It describes the degree to which the source code of a program has been tested.

☞ **Various code coverage criteria**

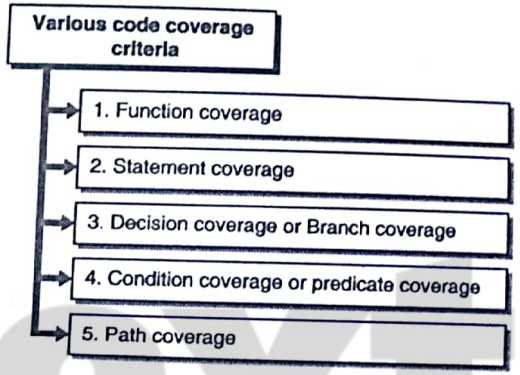


Fig. C11.1 : Various code coverage criteria

➜ **1. Function coverage**

It tests whether each function or subroutine in the program has been called.

Function Coverage = (Number of Functions Exercised/Total number of functions in program)* 100

➜ **2. Statement coverage**

- It tests whether each statement in the program has been executed or not.
- Executes all statements at least once

Statement Coverage = (Number of Statements Exercised/Total number of executable statements in program) * 100

Example :

Assume, program has 100 statements
Tests exercise 87 statements then,
Statement coverage = 87%

Example :

Statement No.	Program
1.	Read(A)
2.	If A > 5 Then
3.	B = A + 1
4.	End-If
5.	Print B

Test Case Id	Input	Expected Output
1	8	9

In this example, as all 5 statements are 'covered' by this test case, we have achieved 100% *statement coverage*

➜ 3. **Decision coverage or Branch coverage**

Has every edge in the program been executed? Such as in IF and CASE statements.

Executes each decision direction at least once.

Decision Coverage= (Number of Decision outcomes Exercised/Total number of Decision outcomes in program)* 100

Example

Assume, program has 100 decision outcomes

Tests exercise 60 decision outcomes then,

Decision coverage = 60%

Example

Statement No.	Program
1	Read(A)
2	If A < 10 or A > 20 Then
3	B = A + 1
4	End-If
5	Print B

Test Case Id	Input	Expected Output
1	8	9
2	22	23

➜ 4. **Condition coverage or predicate coverage**

- Has each Boolean sub-expression evaluated both to true and false? This does not imply decision coverage.
- For example, when there is an OR condition, if first condition is true, the second is not executed. Similarly, in AND condition if the first part is found false, then the second part is not at all evaluated. For these reasons, it is necessary to test each Boolean expression and have test cases to test the true as well as false paths.

Condition Coverage= (Total decisions exercised/Total number of decision in program) * 100

Example 1

A = 10
B = 15
If (A > 3) or (A < B) Then
B = X + Y
End If

Example 2

While (A > 0) and (Not EOF) Do
Read (X)
A = A - 1
End-While-Do

➜ 5. **Path coverage**

- It involves splitting the program into a number of distinct paths. A program can start and take any of the paths to reach the end.
- It executes all possible combinations of condition outcomes in each decision.

Path Coverage= (Total Paths exercised/Total number of executable paths in program) * 100

Example

1. Insert the card
2. IF card is valid THEN
3. display "Enter PIN"
4. IF PIN is valid THEN
5. select AccountType
6. ELSE
7. display "Invalid PIN"
8. ELSE
9. Reject card
10. END

Various paths possible are:

Path 1: 1, 2, 8, 9, 10

Path 2: 1, 2, 3, 4, 6, 7, 10

Path 3: 1, 2, 3, 4, 5, 10

Therefore, minimum path coverage tests to be achieved = 3 since there are three paths.

Example : Consider the following function :

```
int grt(int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0))
    {
        z = x;
    }
    return z;
}
```

- If the function 'grt' was called at least once, then *function coverage* is satisfied.
- *Statement coverage* for 'grt' function is satisfied if every line in the function is executed.
- The grt (1,1) and grt (1,0) will satisfy *decision coverage*, as in the first case the `if` condition is TRUE and in the later it is FALSE.
- *Condition coverage* can be satisfied with tests that call grt (1,1), grt (1,0) and grt (0,0). These are necessary as in the first two cases (x>0) evaluates to TRUE while in the third it evaluates to FALSE. At the same time, the first case makes (y>0) TRUE and in the later two cases it is FALSE.

11.8.2 Code Complexity Testing

Q. Define the terms : Code complexity testing (Cyclomatic complexity)

Cyclomatic Complexity (CC) testing is a process that measures the complexity of a program. In this process, a program is represented through a flowchart – it is then converted to a flow graph containing only the nodes and edges. Steps to convert the flow chart into flow graph are listed down :

Step 1 : Identify the predicates (decision points) in the flow chart.
Step 2 : Ensure that the predicates are simple else break up a condition into simple predicates.

Step 3 : Combine all the sequential statements into a single node. The reason is that all these statements get executed once started.

Step 4 : If a set of sequential statements is followed by a simple predicate, combine all the sequential statements and the predicate check into one node and have two edges emanating from this one node. Such nodes with two edges emanating from them are called predicate nodes.

Step 5 : Make sure that all the edges terminate at some node. Add a node to represent all the sets of sequential statements at the end of the program.

Calculating CC :

- CC = P +1 // where P is Number of Predicates
- CC = E-N+2 // where E is Number of Edges & N is Number of Nodes
- CC = Count of Regions in Flowgraph

Example 1 :

```
Void check()
{ if (ch>5)}
        flg=1
else if(ch=5)
        flg=flg-5
else if(ch>2)
        flg-flg-2
        else
        flg=0
}
```

Solution :

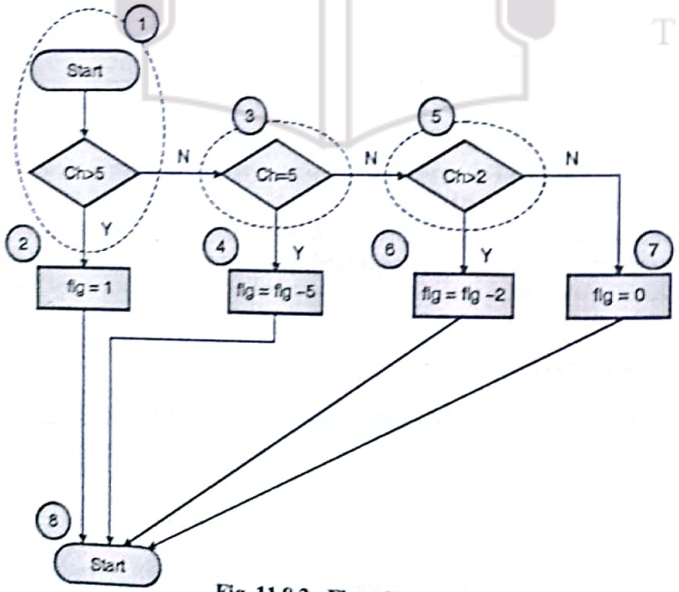


Fig. 11.8.3 : Flow Chart

CC = P + 1 = 3 + 1 = 4

CC = E – N + 2 = 10 – 8 + 2 = 4

CC = Count of Regions = 4

Number of independent paths to be tested using Path coverage are :

Path 1 : 1-2-8

Path 2 : 1-3-4-8

Path 3 : 1-3-5-6-8

Path 4 : 1-3-5-7-8

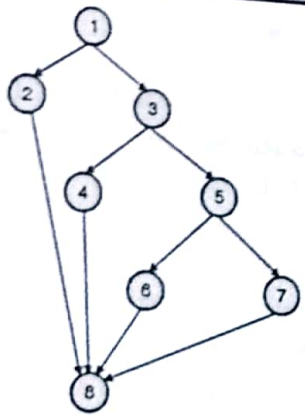


Fig. 11.8.4 : Flow graph

Test Case

TestCase Id	TestCase Condition	Input	Expected Output
TC_01	To validate path 1-2-8	ch=7	flg=1
TC_02	To validate path 1-3-4-8	ch=5	flg=flg-5
TC_03	To validate path 1-3-5-6-8	ch=3	flg=flg-2
TC_04	To validate path 1-3-5-7-8	ch=0	flg=0

Example 2

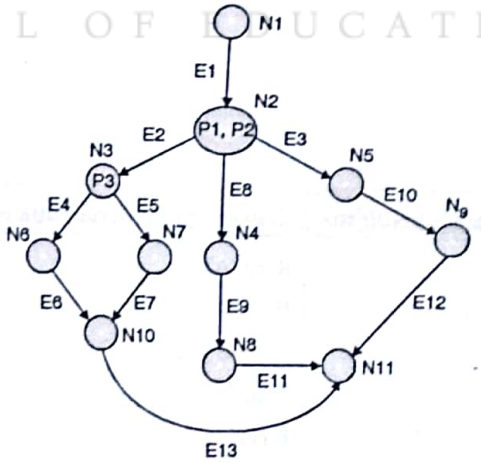


Fig. 11.8.5

CC = P + 1 = 3 + 1 = 4

CC = E – N + 2 = 13 – 11 + 2 = 2 + 2 = 4

CC = Count of Regions = 4

Example 3 :

1. Insert the card
2. IF card is valid THEN
3. display "Enter PIN"
4. IF PIN is valid THEN
5. select AccountType
6. ELSE
7. display "Invalid PIN"
8. ELSE
9. Reject card
10. END

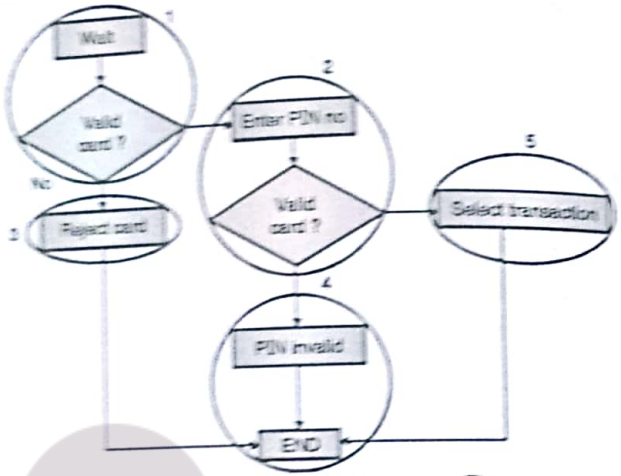


Fig. 11.8.6

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are : 10

Number of Conditions (predicates) to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are:

Path 1: 1 3 6 7
Path 2: 1 2 4 6 7
Path 3: 1 2 5 6 7

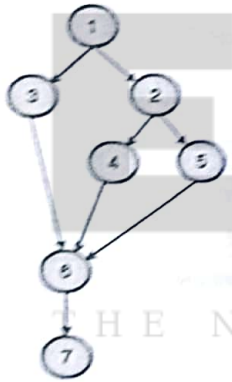


Fig. 11.8.7

Example 4 :

Program with complex predicates	Converted to Program with simple predicates

Read (A)
If A > 0 and A < 5 Then
Print "A"
End If
	Read (A)
	If A > 0 Then
	If A < 5 Then
	Print "A"
	End If
	End If

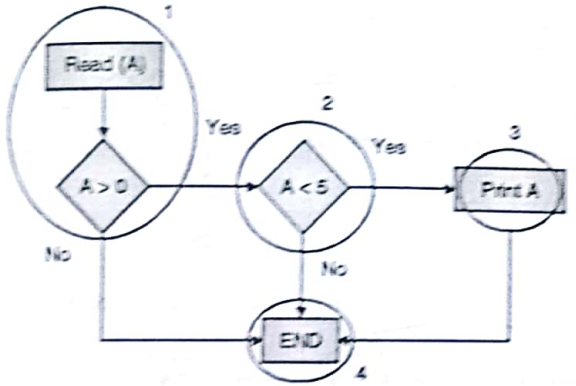


Fig. 11.8.8

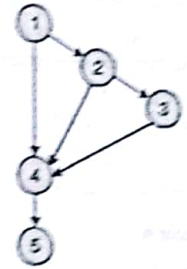


Fig. 11.8.9

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are : 6

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are : 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are:

Path 1: 1 4 5
Path 2: 1 2 4 5
Path 3: 1 2 3 4 5

Example 5

1. Read A
2. Read B
3. IF A > 0 THEN
4. IF B = 0 THEN
5. Print "No values"
6. ELSE
7. Print B
8. IF A > 21 THEN
9. Print A
10. ENDIF
11. ENDIF
12. ENDIF

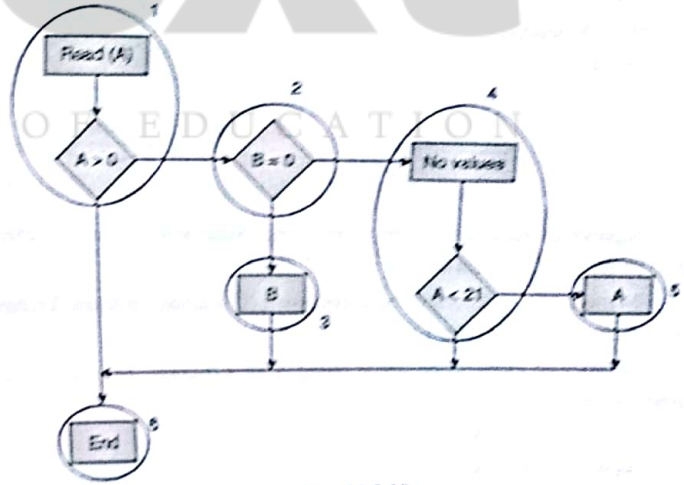


Fig. 11.8.10

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are: 12

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 6 7

Path 2: 1 2 3 6 7

Path 3: 1 2 4 6 7

Path 4; 1 2 4 5 6 7

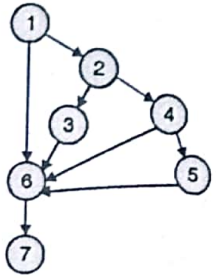


Fig. 11.8.11

Example 6

1. Read A
2. Read B
3. IF A < 0 THEN
4. Print "A negative"
5. ELSE
6. Print "A positive"
7. ENDIF
8. IF B < 0 THEN
9. Print "B negative"
10. ELSE
11. Print "B positive"
12. ENDIF

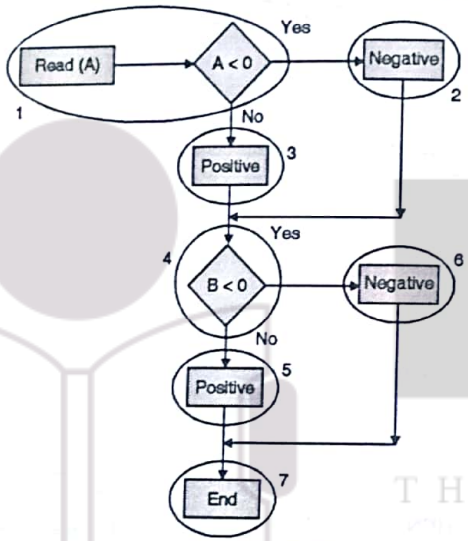


Fig. 11.8.12

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are: 12

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 4 5 7 8

Path 2: 1 3 4 6 7 8

Path 3: 1 2 4 5 7 8

Path 4; 1 2 4 6 7 8

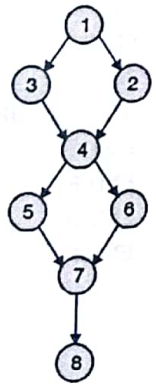


Fig. 11.8.13

Example 7 : Check the cyclometric complexity for a program of adding 100 integers. It should also check for valid boundaries and valid values. Design the Test Cases for such code.

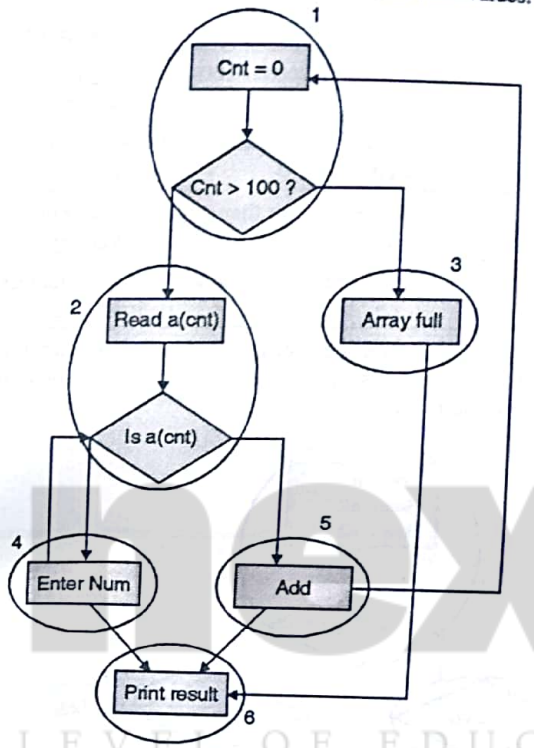


Fig. 11.8.14

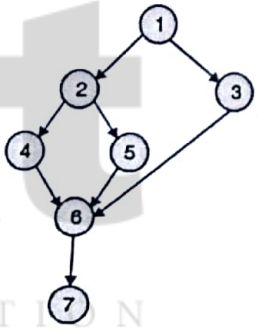


Fig. 11.8.15

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are: 8

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 6 7

Path 2: 1 2 4 6 7

Path 3: 1 2 5 6 7

Test description

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
No. of values in array must be 100 integers to perform addition	= 100 numbers	< 100 numbers > 100 numbers Non digit	100 numbers	99 numbers 101 numbers

Test case

Test Case Id	Conditions	Input	Expected Output
1	No. of values in array must be 100 integers to perform addition	Values till count =100	Addition of Numbers
2		Values less than count =100	Accept more numbers
3		Values greater than count =100	Array Full Warning
4	Non digits not acceptable	Any character or symbol	Enter number warning

Example 8 : A program reads three integer values as three sides of a triangle. The program prints a message indicating that whether the triangle is right angle triangle or equilateral triangle. Draw the flow graph, calculate cyclometric complexity.

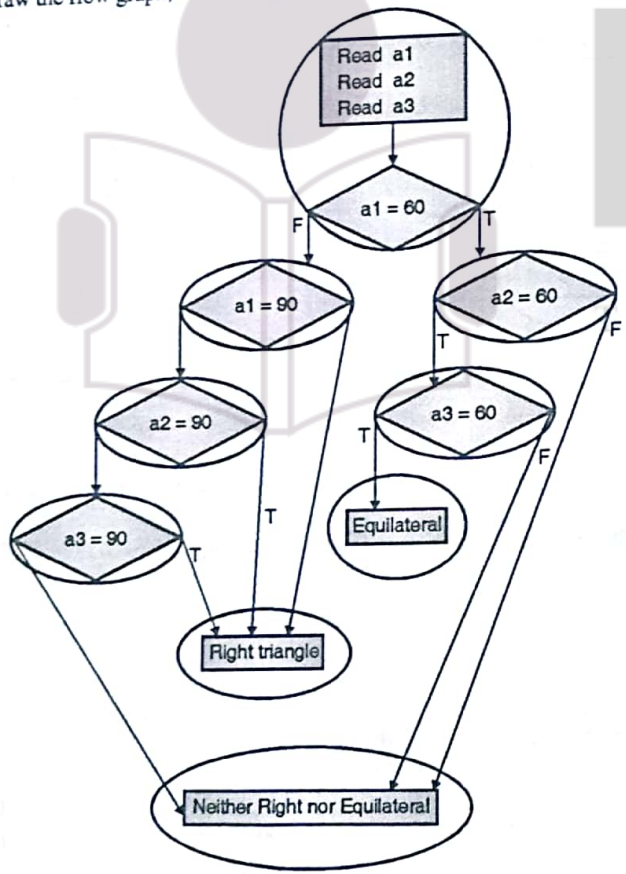


Fig. 11.8.16

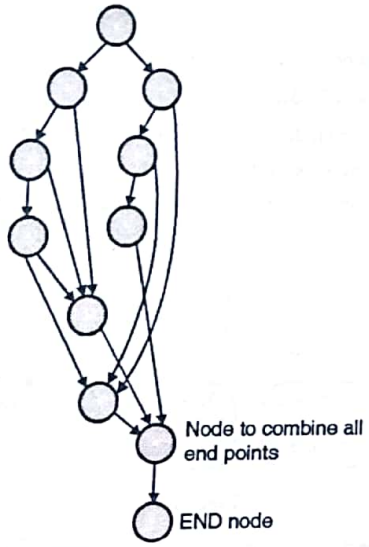


Fig. 11.8.17

Syllabus Topic : Functional / Black Box Testing

11.9 Functional (Black Box) Testing

- It is also called as '*Behavioral testing*' because it only checks the behavior of the system for certain inputs and doesn't think about the internal coding.
- It is concerned with testing the functions of the system. Therefore, it is also called as *Functional Testing*.
  - Black Box Testing involves only the observation of the output for certain input values, and there is no attempt to analyze the programming code that enables to arrive at those outputs. Therefore, it is also termed *data-driven, input/output driven* or *requirements-based testing*.

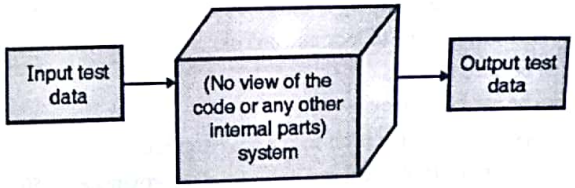


Fig. 11.9.1 : Black Box Testing

Advantages of Blackbox Testing

- Tester does not need any specific programming language knowledge.
- Black-box testing is about checking the data i.e. output data for specific input data so tester need not have any logical knowledge. He just has to check that the outputs are as expected by the user.

- Black-box tester works from the point of view of the user, not the designer.
- Black-box test cases can be designed as soon as the requirement specifications are complete.

Disadvantages of Blackbox Testing

- The tests are sometimes redundant if the software designer has already run a test case.
- The test cases are difficult to design.
- Testing every possible input is not possible because it would take a lot of time and therefore, many of the program paths go untested.

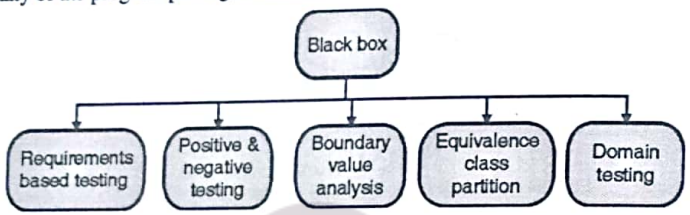


Fig. 11.9.2

11.9.1 Requirements based Testing

- It deals with validating the requirements given in the SRS of the software system.
- Actually, not all the requirements are stated by the users in the SRS - The requirements that are stated explicitly by the users in the SRS are known as ***explicit requirements*** and the requirements that are not stated in the SRS but are assumed to be incorporated in the system are known as ***implicit requirements***.
- This testing ensures that the requirements documented in SRS are consistent, correct, complete and testable.
- This testing also allows combining both explicit and implicit requirements and documenting it together in a single document known as *Test Requirements Specification*.

Example : A black-box test case can be derived from the following requirements for a 'Lock and Key' :

- Inserting key numbered 4321 and turning it clockwise should facilitate locking.
- Inserting key numbered 4321 and turning it anti clockwise should facilitate unlocking.
- Only key number 4321 can be used to lock and unlock.
- No other object can be used to lock.
- No other object can be used to unlock.
- The lock must not open even when it is hit with a heavy object.
- The lock and key must be made of metal and must weigh approximately 150 grams.

11.9.2 Positive and Negative Testing

Positive testing

- It is performed to prove that a given product exactly does what it is expected to do.
- The need of this type of testing is to verify the known test conditions.
- When a test case tests the software to obtain a set of expected output, it is called as positive test case.

- A positive test is done to confirm that the product works as per specification and user expectations.
- It also tests that whether the product gives an error or not when it is expected to give an error. This is also a process of positive testing.

Negative testing

- It is performed to prove that the product does not fail when an unexpected input is given.
- The need of this type of testing is to break the system with unknown inputs because it is important to be aware of the negative situations that may occur at the end-user level and so that the application can be tested and made error proof before delivery to the client.
- A negative test is about testing a product for not delivering an error when it is expected to or delivering an error when it is not expected to.

Example

Sr. No	Positive Testing for 'Lock and Key'	Negative Testing for 'Lock and Key'
1.	Inserting key numbered 4321 and turning it clockwise should facilitate locking.	Insert some other lock's key and turning it clockwise or anti-clock wise should **not** change the current status of the lock i.e. if the lock is already locked then it must not be unlocked using some other key's lock and if it is unlocked then it must not be locked.
2.	Inserting key numbered 4321 and turning it anti clockwise should facilitate unlocking.	Try a thin piece of wire and turning it clockwise or anti-clock wise should **not** change the current status of the lock.
3.	Inserting a hairpin and turning it clockwise or anti-clockwise should have no change in the current status of the lock.	Hitting with a stone should **not** unlock the Lock.

11.9.3 Boundary Value Analysis

Q. Define the terms : BVA (Boundary Value Analysis).

- Most of the defects in software products hover around the conditions and boundaries. By boundaries, we mean 'limits' of values of the various variables. Few errors while defining limits are :
  - Programmer's tentativeness in using the right comparison operator, for example, whether to use the <= operator or < operator when trying to make comparisons.
  - Confusion caused by the availability of multiple ways to implement loops and condition checking.
  - The requirements are not clearly understood, especially around the boundaries thus causing even the correctly coded program to not perform the correct way.
- BVA is useful in catching defects that happen at boundaries.

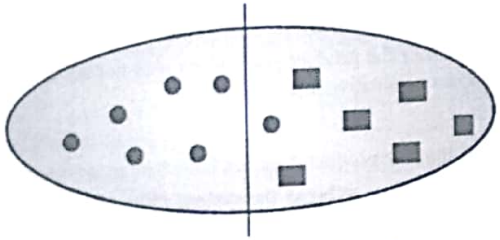


Fig. 11.9.3 : A defect at the boundary

Example : Consider a Grocery store that offers different pricing for people buying in different quantities and this pricing is different in different slabs.

No. Of Units bought	Price per Unit
First five units (i.e. from 1 to 5 units)	Rs. 5.00
Next five units (i.e. from 6 to 10 units)	Rs. 3.00
Next five units (i.e. from 11 to 15 units)	Rs. 2.00

The above table describes that :

- if you buy 4 units then it will cost 4 * 5 = Rs. 20
- if you buy 7 units, then for first 5 units, it cost 5*5=R.25 and for next 2 units it will cost 2*3 =Rs. 6 and therefore 7 units in all costs Rs.25 + Rs.6 + Rs.6 = Rs.31
- if you buy 13 units then for first five units it will cost 5*5=Rs25, next 5 units will cost 5*3=Rs.15 and next 3 units will cost 3*2=Rs.6 and therefore 13 units in all costs Rs.25 + Rs.15 + Rs.6 = Rs. 46
- The most possible defects in such ranges arise around the boundaries. In the above example, the defects may arise while buying 4, 5, 6, 7, 9, 10, 11, 14, 15 units.

11.9.4 Equivalence Class Partition

Q. Define the terms : ECP (Equivalence Class Partition).

- This technique identifies a small set of representative input values that produce as many different output conditions as possible.
- This reduces the number of permutations and combinations of input, output values used for testing. Thereby increasing the coverage and reducing the effort involved in testing.
- The set of input values that generate one single expected output is called a partition. When the behaviour of the software is the same for a set of values then the set is termed as an equivalence class or a partition.

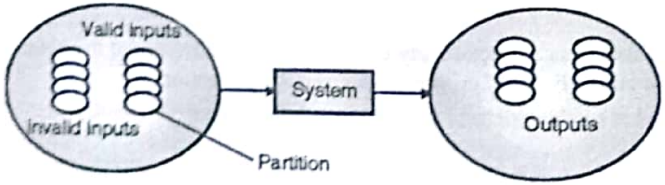


Fig. 11.9.4 : Equivalence class partition

- Testing by this technique involves :
  - Identifying all partitions for the complete set of input, output values for a product.
  - Picking up one member value from each partition for testing to maximize complete coverage.

Example 1 : Consider an example of an insurance company that has the following premium rates based on the age group. This life insurance company has base premium of Rs.1.00 for all ages but the additional monthly premium differs based on their age groups. Therefore, the total premium paid by a person is calculated as base premium + additional monthly premium.

Age group	Additional premium
Under 30	Rs. 2.00
30 - 75	Rs. 4.00
Above 75	Rs. 8.00

Therefore, the *equivalence class partitions* obtained from the above example is based on the age criteria are as given below;

- Below 30 years – valid input
- Between 30 and 75 – valid input
- Above 75 – valid input
- Age as 0 – invalid input
- Negative age – invalid input
- Age in three digit number – valid input

Example 2 : Illustrates BVA and ECP : consider the below form;

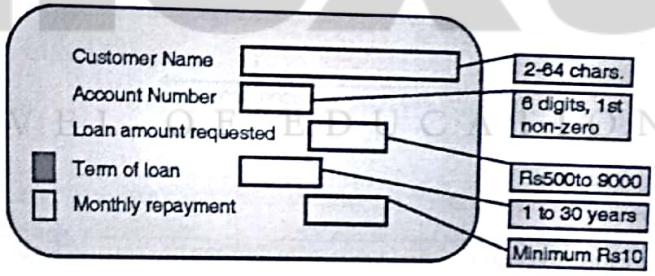
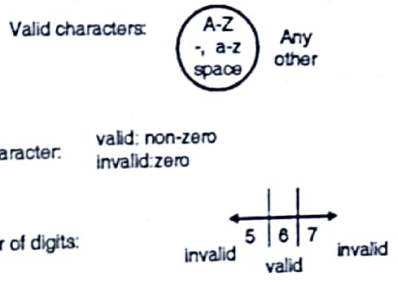


Fig. 11.9.5

The boundaries for different fields are as follows;

1. **Customer Name**



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Customer Name	2 to 64 characters	< 2 chars > 64 chars Invalid characters	2 chars 64 chars	1 char 65 char 0 char

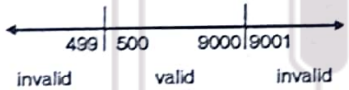
2. **Account Number**

First character: valid: non-zero
invalid:zero

Number of digits:

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Account Number	6 digits First non zero	< 6 digits > 6 digits First digit zero Non digit	100000 999999	5 digits 7 digits 0 digit

3. **Loan Amount**



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Loan Amount	Rs. 500 – 9000	< 500 > 9000 0 Non numeric null	500 9000	499 9001

4. **Term of Loan**

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Term of Loan	1-30 years	< 1 year > 1 year Non digit	1 30	0 31

5. **Monthly Repayment**

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Monthly Repayment	Rs. 10 or more	< 10 Non digit	10	9

Advantages of ECP

- ECP reduces the huge (infinite) set of possible test cases into much smaller, but still equally effective, set.
- This technique allows in achieving good code coverage with very small number of test cases. Say, if there is an error in one value in a partition, then it can affect all the values of that specific partition.

11.9.5 Domain Testing

- This technique is purely based on domain knowledge and expertise in the domain of application. This approach needs understanding of day-to-day business activities for which the software is written.
- The testers for performing this type of testing are selected based on their in-depth knowledge of the business domain. Sometimes, it is easier to hire the testers from the domain area and train them in software, rather than take software professionals and train them in business domain.
- Domain testing is the ability to design and execute test cases that relate to the people who will buy and use the software. It is also called as business vertical testing.

11.9.6 Cause Effect Graphing

- It is a systematic approach to selecting a high-yield set of test cases that explore combinations of input conditions.
- It is a rigorous method for transforming a natural language specification into a formal language specification and exposes incompleteness and ambiguities in the specification.
- This establishes a relation between the causes and effects.
- The causes are the inputs and the effects are the outputs.
- These causes and effects are represented using basic symbols called edges and nodes. These basic symbols represent interdependency between the inputs and outputs.
- The Fig. 11.9.6 depicts various relations between the causes and effects.

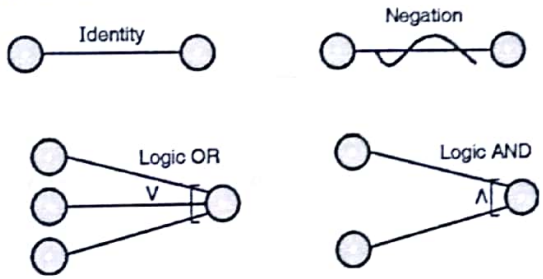


Fig. 11.9.6 : Symbols used in cause-effect graph

Methodology to draw cause effect graph

Step 1 : Decompose the system based on their functions.

Step 2 : Identify the causes also called as conditions

Step 3 : Identify the effects also called as output conditions or actions on the basis of certain conditions

Step 4 : Establish the graph of relations between causes and effects.

Step 5 : Complete the graph by adding the constraints between the causes and effects.

Step 6 : Convert the graph to a decision table and assign true in sequence to all effects

Step 7 : the columns in the decision able are converted into test cases.

Example : Cause-effect graph for 'Employees are given bonus depending upon their appointment status'.

Condition/Causes	Action/Effects
Administrative staff - Permanent	Bonus of 2 months salary
Administrative staff - Temporary	Bonus of 1 month salary
Workers - Permanent	Bonus of 1 month salary
Workers - Temporary	Bonus of ½ month salary

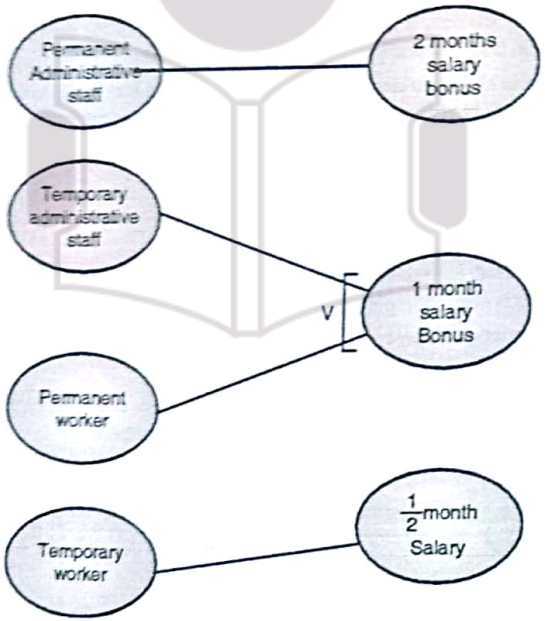


Fig. 11.9.7 : Cause-Effect graph for Employee Bonus

Example : Cause-effect graph for 'Discounts depending upon the size of order'.

Condition	Action
Order size: Over Rs. 5000	5% discount from invoice total
Rs. 2000 to Rs. 5000	3% discount from invoice total
Below Rs. 2000	Pay total invoice amount

Note that the discounts are provided only for 15 days.

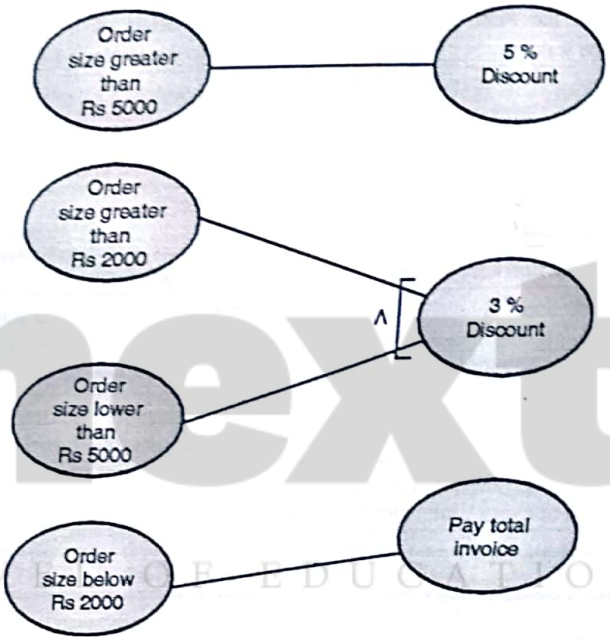


Fig. 11.9.8 : Cause-Effect graph for Discounts

Advantages

- Cause-effect graphs are very useful in the business problems where there are more than one conditions (causes).
- Cause-effect graphs also identify the critical decision processes i.e. they select the effects corresponding to causes.
- Cause-effect graphs describe all the data used in decision making so that the system can be designed to produce the data properly.

Drawbacks

- Cause-effect graphs for a complex system having many combinations of conditions will look very cumbersome.
- Large numbers of branches that represent various combinations of conditions make it difficult for the analysts to understand the formulation of some specific decisions.

11.10 Black Box vs. White Box

Q. Differentiate: whitebox and blackbox testing

Sr. No.	Black-box (Functional) Testing	White-box (Structural) Testing
1	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester
2	Knowledge of Programming language is not required	Deep Knowledge of programming language is required
3	Implementation knowledge is not required	Implementation knowledge is required
4	The tester examines the application's external functional behavior and GUI features.	The tester has to correct the code also.
5	done by independent Software Testers	done by developers
6	Black box can be applicable at all levels but dominates at higher levels of testing.	White box dominates at lower levels.
7	Requirement specification is the basis for conducting black box test	Detail design specification is the basis for conducting white-box test
8	in black box testing, sample test cases are selected and applied every time when required.	We have to write large quantity of test cases for white-box testing.

Syllabus Topic : Test Plan

11.11 Test Plan

- Test Plan tells **"What to Test"**
- The purpose of the Test Plan document is to :
  - Specify the approach to test the product
  - It specifies the deliverables extracted from the Test Approach.
  - Breakdown the product and identify features to be tested.
  - Specify the test strategies
  - Indicate the test tools to be used.
  - List the resource and scheduling plans.
  - Indicate the contact persons responsible for various areas of the project
  - Identify risks and contingency plans.
  - Specify bug management procedures for the project.
  - Specify criteria for acceptance of the product.

11.11.1 Test Plan Template

(Name of the Product)

Prepared by :

(Names of Preparers)

(Date)

Table of Contents :

INTRODUCTION :

1. TEST PLAN OBJECTIVES

- Describe the objectives supported by the Master Test Plan, eg., defining tasks and responsibilities, vehicle for communication, document to be used as a service level agreement, etc.
- [... A primary objective of testing application systems is to: assure that the system meets the full requirements, including quality requirements. At the end of the project development cycle, the user should find that the project has met or exceeded all of their expectations as detailed in the requirements.
- The secondary objective of testing application systems will be to: identify and expose all issues and associated risks, communicate all known issues to the project team, and ensure that all issues are addressed in an appropriate matter before release. ...]

2. SCOPE

2.1 Breaking down the product into features

2.2 Prioritizing the features based upon:

Features that are new and have higher susceptibility to defects

Features whose failures can have adverse business impact

Features that are difficult to test

Features that are extension of earlier features that have been defect prone

2.3 Deciding features to be tested

2.4 Deciding features not to be tested

3. TEST STRATEGY (Tells "How to Test" or defines the approach to test.) :

- what type of testing types need to be carried out, levels of testing techniques, method and tools to be used
- This includes what type of testing to be used :

3.1 System Test

3.2 Performance Test

3.3 Security Test

3.4 Functionality Test

3.5 Stress and Load Test

3.6 Recovery Test

3.7 Documentation Test

3.8 Beta Test

3.9 User Acceptance Test

4. DEPENDENCIES

- Dependencies may be categorized as external and internal based on which WBS can be formed. External dependencies are beyond the control of the Test leader. Some common external dependencies are:
- Availability of product from developers
- Hiring
- Training
- Acquisition of hardware/software required for training
- Internal dependencies are fully within the control of the tester. For example; Completing the test specification, Designing the test cases, Executing the tests

5. RISKS

5.1 Schedule Risks

Testing scheduling completely depends on the development schedule. Thus, it becomes difficult for the testing team to line-up resources i.e. people and tools in right manner at right time. This testing scheduling risk is even more severe in cases where a testing team is shared across multiple projects.

5.2 Technical

- Fixing some of the defects could lead to changes in the architecture and design. Carrying out such changes into the cycle may be expensive or even impossible. Once the developers fix the defects, the testing team would have even lesser time to complete the testing and is under even greater pressure i.e. **causing insufficient time for testing.**
- Certain defects are show stoppers may prevent the testing team to proceed further with testing, until development fixes such **show stopper defects.**

5.3 Management Risks

- Management has to take lot of efforts in hiring skilled and motivated people and they must also be available at required time.
- Management may fail in arranging the test automation tools, also manual testing might be error prone and labour intensive. Also, the test automation tools are expensive. And, the organization may face the risk of not being able to afford a test automation tool. Such risks lead to less effective and efficient testing.

5.4 Requirements Risks

When the requirements specified are not clearly documented, there is ambiguity in how to understand the results of a test. This could result in wrong defects being reported or in the real defects being missed out. This in turn, results in unnecessary and wasted cycles of communication between the development and testing teams and ultimately the loss of time.

Risk management involves

- Identifying the possible risks through use of checklists, organizational history and metrics, and informal network across the industry.
- Quantifying the risks upon probability of the risk and the impact of the risk. For example, show stopper defects have low probability but high impact.

- Planning how to mitigate the risks: It deals with identifying alternative strategies to reduce the risk events.
- Responding to risks when they become a reality.

6. ESTIMATIONS

6.1 Size Estimates quantifies the actual amount of testing that needs to be done.

Size of the product can be measured as follows :

- Lines Of Code (LOC) depends on the language, style of programming, compactness of programming, and so on.
- A function point is the application functions classified as inputs, outputs, interfaces, external data files and enquiries.
- Number of screens, reports or transactions - Each of these can be classified as simple, medium or complex.

Size of the testing can be measured as follows:

- Number of test cases
- Number of test scenarios

6.2 Effort Estimates

Factors that derive effort estimates:

- **Productivity Data :** speed at which various activities of testing can be carried out and this estimation is done based upon the historical data available in the organization.
- **Reuse opportunities :** Re-using concept ultimately reduces the efforts needed, Therefore, if the test architecture has been designed by concerning the reuse idea, then the effort required to do the same task once again obviously comes down.
- **Size estimate** also derives the estimation of efforts required. Say, for example, if some of the test cases or some part of the test cases can be reused from existing test cases, then the efforts involved in developing these would be near to zero. On the other hand, if a given test case is to be developed fully from scratch, it is obvious that the efforts must be put again which would increase the size of the test case i.e. divided by productivity.
- **Effort estimate** is measured in persons required per days, person required in number of months or person required in number of years. The effort estimate is then used to achieve the schedule estimation.

6.3 Schedule Estimates

It includes :

- Determining the external and internal dependencies among the activities
- Arranging the activities in a sequence based on the expected duration as well as on the dependencies
- Determining the time required for each of the WBS activities taking into account the above two factors.
- Monitoring and controlling the progress in terms of time and effort.
- Modifying and balancing the schedules and resources as necessary.
- Determine the time required to do perform each testing task. Also, specify the schedule for each testing task and test milestone. For each testing resource such as facilities, tools, and staff, specify its periods where it is necessary to use.

7. **TEST DELIVERABLES**

These are the documents that come out of the test cycle activity. The deliverables include :

- Test plan itself (Master test plan and various other test plans of the project)
- Test case design specifications
- Test logs produced by running the tests
- Test case coverage reports
- Test case results
- Test status reports
- Test summary reports

8. **RESPONSIBILITIES (Tells "Who will do the test")**

It includes identifying responsibilities, staffing and training them. The different role definitions should :

- Ensure there is clear accountability for a given task, so that each person knows what he or she has to do.
- Clearly list the responsibilities for various functions to various people, so that everyone knows how his or her work fits into the entire project
- Complement each other, ensuring no one steps on an others' toes
- Supplement each other so that no task is left unassigned.
- Staffing is done based on estimation of effort involved and the availability of time for release.

9. **HARDWARE AND SOFTWARE REQUIREMENTS**

Some of the hardware and software factors need to be considered are:

- Machine configuration (RAM, processor, disk and so on) needed to run the product under test.
- Compilers, test data generators, configuration management tools.
- Special requirements for running machine-intensive tests such as load or stress tests.

10. **ENVIRONMENT REQUIREMENTS**

- Client side operating system
- Client side h/w and s/w req.

11. **DEFECT RECORDING**

Document the procedures to log a defect.

12. **APPROVALS**

- Names and titles of all persons who should approve this plan. Provide space for the signatures and dates.
- Name (In Capital Letters) Signature Date Of Approval

1.

2.

3.

4.

11.11.2 Test Plan Examples

Example 1 : Write a test plan for a web based MSEB application system which provides the facilities like bill payment, customer grievance cell in support of complaints and feed back. Your plan should include the test documents and test strategies. (Make suitable assumptions if required)

Solution :

MSED provides electricity and keeps track of the usage of electricity in the whole Maharashtra. But their work is spread ona very wide area and the number of users are also in crores. Therefore, MSEB wants to automate the following functionalities. The functionality should be made available on-line (i.e. web-based)

Functionalities

(1) Maintaining the bill payment of customers
(2) Maintaining the user records.
(3) Maintaining customer grievance
(4) MSEB is using third party payments gateways of different partners like ICICI, HDFC etc.

1. **Test plan Objectives**

Test plan objectives for new web application

(i) Define the activities required to develop and test the system.
(ii) Communicate the test strategy to all responsible parties the System.
(iii) Define deliverables.
(iv) Communicate the various Dependencies and Risks to all stakeholders.

2. **Scope**

(a) **Data entry :** The new functionality of MSEB should allow the system administrator of the proposed system to enter the items of different Departments (billing dept., feedback dept., grievance dept., customer records dept. and so on), management should be able to view details of bills and address information of customers. The system should be user friendly and will provide error message to help direct the administrator through various options.

(b) **Security :** Each Customer will need a User id and password to login to the system. The system will allow the Customer to change his/her password whenever he/she wants to. Similarly as customers are paying money through third party, tight security should be provided to that interface.

3. **Test Strategy**

The test strategy consists of different types of tests that will fully exercise the online MSEB system.

(i) **System test :** The system test will focus on the behavior of the MSEB system. The system tests will test the integrated system and verifies that it meets all the customer requirements defined in the SRS.

(ii) **Performance Test :** Performance test will be conducted to ensure that the MSEB system's response time meets the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress and load.

(iii) **Security Test :** The tests will verify that unauthorized user access to confidential data is prevented.

(iv) **Configuration and Compatibility Test :** This is very essential tests required for web application. It determines how system performs in a particular hardware, software, operating system, network etc. environment.

(v) **Usability Test :** This test will determine how the system is user friendly. In MSEB system Customer and the employees of the MSEB offices are the ultimate users, so they should be able to handle all functionality of the system very easily.

(vi) **Automated Test :** A suite of automated tests will be developed to test the basic functionality of the online MSEB system and perform regression testing on areas of the system that previously had critical/major defects.

(vii) **Beta Test :** The MSEB staff will perform beta tests on the new online system and will report any defects they find to the test team and the corresponding development team. This will subject to tests that could not be performed in developer's test environment.

(viii) **User Acceptance Test :** Once the online system is ready for implementation, the MSEB authority will perform User Acceptance Testing. The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

4. Environment Requirements

(a) Workstation

1. Pentium Processor
2. 256 RAM
3. 40 GB Hard Disk
4. Windows XP/NT/2000/Linux/Mac
5. LAN Network
6. Any current trend browsers

(b) Server and Client Side Software

1. Web Server - Apache
2. Programming language – Java, JSP, Servlets.
3. Database - Oracle

5. Functions to be tested

The following is a list of functions that will be tested

(i) Add/update Customer information
(ii) Add/update bill pyment information
(iii) Add/update customer grievance and feedback
(iv) Add/update details from different departments
(v) Search for customers or areas or bill balances
(vi) Security features
(vii) Error messages
(viii) Payment through third party
(ix) Reports of MSEB

6. Resources

The test team will consists of :

(i) A project Manager
(ii) A test lead
(iii) Test team
(iv) The MSEB Manager
(v) MSEB employees
(vi) MSEB Customers

Example 2 : Peoples-Bazaar is a famous shopping mall spread globally. Shopping mall wants to automate the following functionality.

1) Maintaining user details, giving proper authentication
2) Maintaining items under different Departments, like music, clothing, etc.
3) Maintaining online customer order.
4) Providing on-line payment facility towards order.

'Peoples-Bazaar' is using third party payment gateways of different partners like ICICI, HDFC etc.

Design test plan, define scope, testing strategies, coverage, environment and other design details.

Solution :

1. Test plan Objectives

Test plan objectives for new web application

(i) Define the activities required to develop and test the system.
(ii) Communicate test strategies to all responsible parties.
(iii) Define deliverables.
(iv) Communicate various Dependencies and Risks to all responsible parties.

2. Scope

(a) **Data entry :** The new functionality of shopping mall should allow Administrator of this proposed system to enter the items of different Departments, view orders of customers. The system should be user friendly and will provide error message to help direct the administrator through various options.

(b) **Security :** Each Customer will need a User id and password to login to the system. The system will allow the Customer to change his/her password whenever he/she wants to. Similarly as customers are paying money through third party, tight security should be provided to that interface.

3. Test Strategy

The test strategy consists of different types of tests that will fully exercise the online shopping mall.

(i) **System test :** The system test will focus on the behavior of the MSEB system. The system tests will test the integrated system and verifies that it meets all the customer requirements defined in the SRS.

(ii) **Performance Test :** Performance test will be conducted to ensure that the shopping mall system's response time meets the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress and load.

(iii) **Security Test :** The tests will verify that unauthorized user access to confidential data is prevented.

(iv) **Configuration and Compatibility Test :** This is very essential tests required for web application. It determines how system performs in a particular hardware, software, operating system, network etc. environment.

(v) **Usability Test :** This test will determine how the system is user friendly. In online shopping mall system Customer is a user, so he/she should be able to handle all functionality of the system very easily.

(vi) **Automated Test :** A suite of automated tests will be developed to test the basic functionality of the online shopping mall system and perform regression testing where necessary.

(vii) **Beta Test :** The shopping mall staff will beta tests the new online system and will report any defects they find. This will subject to tests that could not be performed in our test environment.

(viii) **User Acceptance Test :** Once the online system is ready for implementation, the shopping mall authority will perform User Acceptance Testing. The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

4. **Environment Requirements**

(a) Workstation
   1. Pentium Processor
   2. 256 RAM
   3. 40 GB Hard Disk
   4. Windows XP/NT/2000/Linux/Mac
   5. LAN Network
   6. Any current trend browsers

(b) Server and Client Side Software
   1. Web Server – Internet Information Server
   2. Programming language – ASP
   3. Database - Oracle

5. **Functions to be tested**

The following is a list of functions that will be tested

(i) Add/update Customer information
(ii) Add/update Items from different departments
(iii) Search/lookup Item information
(iv) Security features
(v) Error messages
(vi) Placing order
(vii) Payment through third party
(viii) Reports for shopping mall

6. **Resources**

The test team will consists of:

(i) A project Manager
(ii) A test lead
(iii) Test team
(iv) The shopping Mall Manager
(v) Shopping mall employees

Example 3 : A company has designed its computerized payroll system to handle the following activites.

1. Provide the user with payslip containing basic all allowances, all deductions
2. Store the employee information
3. Security employee information
4. Prints various reports
5. Provide compatibility with mainframe and PCS and LAN.
6. Write a test plan.

Solution :

SAN's has recruited number of employees and the company was wide spread. And it was impossible to maintain the details of all these employees manually. And, also if the manager sitting in any other department needed the details of employees of other department, he had to get it manual documents, or the softcopy in the pendrive or by mail. Therefore, SAN company wanted to automate the following functionalities. The functionality should be made available on-line (i.e. web-based)

Functionalities

1. Maintaining the employee details
2. Secure the employee records.
3. Maintain the pay slips and payments of all the employees.
4. Produce the payments by cuttings and increments or bonus.
5. Provide print reposts of employee details, their payments and information about their leaves and vacation.

1. **Test plan objectives**

Test plan objectives for new web application

(i) Define the activities required to develop and test the system.
(ii) Communicate to all responsible parties the System Test Strategy.
(iii) Define deliverables and responsible parties.
(iv) Communicate to all responsible parties the various Dependencies and Risks.

2. **Scope**

(a) **Data entry :** The new functionality of SAN company should allow the system administrator of the proposed system to enter the items of different Departments (employee personal details, payment details, leave record details, over time details and so on), management should be able to view details of payments, contact info., leave and over time information of employees. The system should be user friendly and will provide error message to help direct the administrator through various options.

(b) **Security :** Each employee will need a User id and password to login to the system. The system will allow the Customer to change his/her password whenever he/she wants to.

3. **Test strategy**

The test strategy consists of different types of tests that will fully exercise the SAN company.

(i) **System test :** The system test will focus on the behavior of the MSEB system. The system tests will test the integrated system and verifies that it meets all the customer requirements defined in the SRS.

(ii) **Performance Test :** Performance test will be conducted to ensure that the SAN system's response time meets the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress and load.

(iii) **Security Test :** Security tests will determine how to secure the new online SAN system is. The tests will verify that unauthorized user access to confidential data is prevented.

(iv) **Configuration and Compatibility Test :** This is very essential tests required for web application. It determines how system performs in a particular hardware, software, operating system, network etc. environment.

(v) **Usability Test :** This test will determine how the system is user friendly. In SAN system Top Management and the employees of the SAN office are the ultimate users, so they should be able to handle all functionality of the system very easily.

(vi) **Automated Test:** A suite of automated tests will be developed to test the basic functionality of the online SAN system and perform regression testing on areas of the system that previously had critical/major defects.

(vii) **Beta Test :** The SAN staff will perform beta tests on the new online system and will report any defects they find to the test team and the corresponding development team. This will subject to tests that could not be performed in developer's test environment.

(viii) **Acceptance Test :** The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

4. **Environment requirements**

(a) Workstation

1. Pentium Processor
2. 256 RAM
3. 40 GB Hard Disk
4. Windows XP/NT/2000/Linux/Mac
5. LAN Network
6. Any current trend browsers

(b) Server and Client Side Software

1. Web Server – Apache
2. Programming language – Java, JSP, Servlets.
3. Database - Oracle

5. **Functions to be tested**

The following is a list of functions that will be tested

(i) Add/update the employee details

(ii) Secure the employee records.

(iii) Add/update the payment details of employees.

(iv) Add/update the employee cuttings and increments or bonus.

(v) Add/update the employee leave, vacation and over time details

(vi) Report generation of employee details, their payments and information about their leaves and vacation.

6. **Resources**

The test team will consists of :

(i) A project Manager

(ii) A test lead

(iii) Test team

(iv) The SAN top management

(v) SAN employees

(vi) SAN Customers

Syllabus Topic : Test-Case Design

11.12 Test Cases

Test case is defined as

- A set of test inputs, execution pre-conditions and expected outputs developed for a particular objective.
- The inputs will be tried to match the actual output with the expected output to verify the correctness of the software.

Contents of a test case

- Test Item : specifies the name of a particular component of the product that is to be tested
- Test case id : it is a unique identifier number given to the test cases.
- Test case - name and description : This clearly states what needs to be tested
- Test pre-condition : it indicates the stae of the system in which it is required to be before executing the test case.
- Input : these are the sequences to be entered by the user – it may be a numeric value, alphabets, symbols, just a key stroke or anything that is listed down there.
- Expected output : It states what should happen when the test case is executed
- Actual output : It indicates whether the test case is behaving in the same manner as expected output or not. It give space to the tester to record unexpected results.

Test Case Designing

1. Test Item
2. Preconditions
3. Test case

testcase id	test case name	test case desc	Test steps			test case status	test status (P/F)	test priority
			Step / Input	Expected Output	Actual Output			

11.12.1 Login Test Case

- test URL : www.yahoo.co.in/login
- Preconditions : Open Web browser and enter the given url in the address bar. Home page must be displayed. All test cases must be executed from this page.

Test case id	Test case name	Test case desc	Test steps			Test case status	Test status (P/F)	Test Priority
			Step / Input	Expected Output	Actual Output			
Login01	Validate Login	To verify that Login name on login page must be greater than 3 characters	Enter login name less than 3 chars (say a) and password and click Submit button	An error message "Login not less than 3 characters" must be displayed		Design		High
			Enter login name less than 3 chars (say ab) and password and click Submit button	An error message "Login not less than 3 characters" must be displayed		Design		High
			Enter login name 3 chars (say abc) and password and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High
Login02	Validate Login	To verify that Login name on login page should not be greater than 10 characters	Enter login name greater than 10 chars (say abcdefghijk) and password and click Submit button	An error message "Login not greater than 10 characters" must be displayed		Design		High
			Enter login name less than 10 chars (say abcdef) and password and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayeld		Design		High

Test case id	Test case name	Test case desc	Test steps			Test case status	Test status (P/F)	Test Priority
			Step / Input	Expected Output	Actual Output			
Login03	Validate Login	To verify that Login name on login page does not take special characters	Enter login name starting with specail chars (!hello) password and click Submit button	An error message "Special chars not allowed in login" must be displayed		Design		High
			Enter login name ending with specail chars (hello$) password and click Submit button	An error message "Special chars not allowed in login" must be displayed		Design		High
			Enter login name with special chars in middle(he&^llo) password and click Submit button	An error message "Special chars not allowed in login" must be displayed		Design		High
Pwd01	Validate Password	To verify that Password on login page must be greater than 6 characters	enter Password less than 6 chars (say a) and Login Name and click Submit button	An error message "Password not less than 6 characters" must be displayed		Design		High
			Enter Password 6 chars (say abcdef) and Login Name and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High
Pwd02	Validate Password	To verify that Password on login page must be less than 10 characters	Enter Password greater than 10 chars (say a) and Login Name and click Submit button	An error message "Password not greater than 10 characters" must be displayed		Design		High
			Enter Password less than 10 chars (say abcdefghi) and Login Name and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High
Pwd03	Validate Password	To verify that Password on login page must be allow special characters	Enter Password with special characters(say !@hi&*P) Login Name and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High

Test case id	Test case name	Test case desc	Test steps			Test case status	Test status (P/F)	Test Priority
			Step / Input	Expected Output	Actual Output			
Link01	Verify Hyperlinks	To Verify the Hyper Links available at left side on login page working or not	Click Home Link	Home Page must be displayed		Design		Low
			Click Sign Up Link	Sign Up page must be displayed		Design		Low
			Click New Users Link	New Users Registration Form must be displayed		Design		Low
			Click Advertise Link	Page with Information and Tariff Plan for Advertisers must be displayed		Design		Low
			Click Contact Us Link	Contact Information page must be displayed		Design		Low
			Click Terms Link	Terms Of the service page must be displayed		Design		Low
Flink01	Verify Hyper links	To Verify the Hyper Links displayed at Footer on login page working or not	Click Home Link	Home Page must be displayed		Design		Low
			Click Sign Up Link	Contact Information page must be displayed		Design		Low
			Click Contact Us Link	Page with Information and Tariff Plan for Advertisers must be displayed		Design		Low
			Click Advertise Link	Terms Of the service page must be displayed		Design		Low
			Click Terms Of Membership Link	Privacy Policy page must be displayed		Design		Low
			Click Privacy Policy Link	Privacy Policy page must be displayed		Design		Low
Lblink01	Verify Hyper links	To Verify the Hyper Links displayed at Login Box on login page working or not	Click NEW USERS Link located in login box	New Users Registration Form must be displayed		Design		Low
			Click New Users(Blue Color) Link located in login box	New Users Registration Form must be displayed		Design		Low
			Click Forget Password Link located in login box	Password Retrieval Page must be displayed		Design		Medium

11.12.2 Email Address Test Case

Guidelines are given below; Try to design the test case template properly.

- REQUIREMENTS : consider a text box It should accept only 10 characters - 20 characters that may include @, _ , . symbols and small alphabets. So, here we should use boundary value analysis and equalence patition technique to write this test case.
- TEST CASE :There are many validation for E-mail here we are defining some of them.
- Email format: vishal.bilgaiyan @ gmail.com

(1) (2) (3) (4)

- We have divided Email ID in 4 parts.
- In first part we have to check invalid condition input spaces, Left blank, use special character, symbols etc.
- In Second part we have to check invalid condition input other specail character [!#$%^&*()_-:;"'?/|\) at the place of @. left this field as blank. input spaces.etc.
- In third part we have to check invalid domain.
- In fourth part we have to check only valid suffix(.com,.co.in,.edu,.info,.org,.in etc)

Valid Email address	Reason
email@domain.com	Valid email
firstname.lastname@domain.com	Email contains dot in the address field
email@subdomain.domain.com	Email contains dot with subdomain
firstname+lastname@domain.com	Plus sign is considered valid character
email@123.123.123.123	Domain is valid IP address
email@[123.123.123.123]	Square bracket around IP address is considered valid
"email"@domain.com	Quotes around email is considered valid
1234567890@domain.com	Digits in address are valid
email@domain-one.com	Dash in domain name is valid
a_b@domain.com	Underscore in the address field is valid
email@domain.name	.name is valid Top Level Domain name
email@domain.co.jp	Dot in Top Level Domain name also considered valid (use co.jp as example here)
firstname-lastname@domain.com	Dash in address field is valid

Invalid Email address	Reason
plainaddress	Missing @ sign and domain
#@%^%#$@#$@#.com	Garbage
@domain.com	Missing username
Joe Smith <email@domain.com>	Encoded html within email is invalid
email.domain.com	Missing @

Invalid Email address	Reason
email@domain@domain.com	Two @ sign
.email@domain.com	Leading dot in address is not allowed
email.@domain.com	Trailing dot in address is not allowed
email..email@domain.com	Multiple dots
あいうえお@domain.com	Unicode char as address
email@domain.com (Joe Smith)	Text followed email is not allowed
email@domain	Missing top level domain (.com/.net/.org/etc)
email@-domain.com	Leading dash in front of domain is invalid
email@domain.web	.web is not a valid top level domain
email@111.222.333.44444	Invalid IP format
email@domain..com	Multiple dot in the domain portion is invalid

11.12.3 Test Case for Calculator

Guidelines are given below; Try to extend the test case template properly.

Test Case Id	Test Case Name
01	Verify all the basic functionality +,-,*,/ .
02	Verify complex functionality like sqrt for both +ve and -ve Numbers
03	Verify by dividing by Zero
04	Verify the expressions like 2+-4, -4+-4, -4-4
05	Verify = button
06	Verify whether pressing AC button clears the screen
07	Verify by multiplying by Zero
08	Verify that multiplication of two negative numbers is positive. (-2*-3 = 6)

11.12.4 Test Case for ATM

Guidelines to test few of the functions that are performed while withdrawing an amount from an ATM are given below; Try to extend the test case template properly.

Test Case Id	Test Case Name	Input	Expected Output
01	Verify that system reads a customer's ATM card	Insert a readable card	Card is accepted System asks for entry of PIN
02	Verify that system rejects an unreadable card	Insert an unreadable card	Card is ejected and system displays an error screen and is ready to start a new session
03	Verify that system accepts customer's PIN	Enter PIN	System displays a menu of transaction types

Test Case Id	Test Case Name	Input	Expected Output
04	Verify that system allows customer to perform a transaction	Perform a transaction	System asks which type of transaction to be performed.
05	System allows multiple transactions in one session	Answer YES	System displays a menu of transaction types
06	Session ends when customer chooses not to do another transaction	Answer NO	System ejects the ATM card and is ready to start a new session
07	System properly handles an invalid PIN	Enter an incorrect PIN and then try a transaction	The Invalid PIN Extension is performed
08	System asks customer to choose type of account to withdraw from	Choose savings account	System displays a screen where you can enter the withdrawal amount.
09	System verifies that customer has sufficient balance to fulfill his request of withdrawal amount	Enter the amount greater than the account balance	System displays an appropriate message and gives the customer an option of choosing to do another transaction or not.
10	Three incorrect re-entries of PIN result in retaining the card, aborting the transaction and locking the account for 48 hours	Enter incorrect PIN for three times	An appropriate message is displayed, card is retained by machine, session is terminated and the account gets automatically locked for 48 hours.

Review Questions

Q. 1. Define the terms :

1. Walk-through
2. Inspection
3. Desk-checking
4. Code Review
5. Code complexity testing (Cyclomatic complexity)
6. BVA (Boundary Value Analysis)
7. ECP (Equivalence Class Partition)
8. Bug life cycle

Q. 2 Differentiate:

1. Static and dynamic testing
2. Integration and System Testing

□□□

Model Question Paper - I

Q. 1 Attempt all(Each of 5 Marks) : **(15 Marks)**

a. Choose the correct answer

1) What is a Software ?

a) Software is a programming language that is used to build an application

b) Software is a set of instructions to acquire inputs and to process them to produce the desired output

c) Software encompasses of only instructions

d) Software wears out as the time passes due to the effects of dust, vibration, temperature extremes and many such environmental problems.

e) Software encompasses of disks, disk drives, display screens, keyboards, printers and chips

2) What are the tasks in requirement engineering ?

a) Inception, elicitation, elaboration, negotiation, specification, validation and management

b) Inspection, elicitation, elaboration, specification, validation and monitoring

c) Elicitation, elaboration, negotiation, specification, validation and management

d) Inception, elicitation, elaboration, specification, validation and monitoring

e) Inception, elicitation, elaboration, negotiation, specification, validation

3) What is project management ?

a) The planning, organizing, monitoring and controlling of all aspects of the project in a continuous process in order to achieve its objectives

b) The planning and application of business and financial models used to control all aspects of the project for the purpose of meeting project objectives

c) The application of knowledge, skills, tools and techniques to project the activities for the purpose of meeting or exceeding stakeholder objectives

d) Both A and C

4) Project schedules can be displayed graphically as :

a) Gantt charts b) Precedence diagrams

c) Milestone charts d) All of the above

5) Which term refers to hiding of information ?

a) Inheritance b) Encapsulation c) Coupling

d) Cohesion e) Messaging

b. Fill in the blanks

a) The _____ model is also called as classic life cycle.

b) CASE is short for _____

c) The main objective of Software testing is to find _____

d) The two categories of requirements are _____ and _____

e) _____ is a framework for the iterative process of planning, tracking and reacting to risk.

c. Answer in one sentence

a) What is meant by coupling ?

b) Define the term Abstraction.

c) What is top down estimating approach based on ?

d) What is the need of Attribute Hiding Factor ?

e) State the three formulae used to determine the cyclomatic complexity.

Q.2 Attempt the following (Any three) **(15 Marks)**

a) What is meant by concurrent development model ?

b) Define legacy software.

c) What is agility ? Explain XP in detail.

d) What is SRS ? Explain need & benefits of SRS.

e) What is Risk Management ? Explain the different stages involved in Risk Management.

Q.3 Attempt the following (Any three) **(15 Marks)**

a) Write a short note on spiral model.

b) What is Test First Development ? State its advantages.

c) Give various approaches for identifying classes. Explain any 2 in brief.

d) Explain COCOMO model in detail with example. Give its advantages.

e) Calculate Cyclomatic Complexity of code which accepts 3 integer values from the user as input and sort them in ascending order. Find various paths and design test cases.

Q.4 Attempt the following (Any three) **(15 Marks)**

a) What is meant by Quality Function Deployment ?

b) Prepare an Activity diagram for booking a flight through Airline Reservation system.

c) Draw an activity network for the project given below :

Activities	Duration	Precedents
A – Selection of hardware	6	-
B – design of software	4	-
C – install hardware	3	-

Activities	Duration	Precedents
D – code an test software	4	A
E – file take on	3	B
F – write user manual	10	-
G – user training	3	E, F
H – install and test the system	2	C, D

d) Explain the concept of Make/Buy decision

e) Explain the McCall's Quality factors.

Q.5 Attempt the following (Any three) **(15 Marks)**

a) What is an elicitation? Discuss the problems that are encountered during elicitation.

b) Draw a sequence diagram for online ordering of home delivery pizza.

c) Explain the various types of testing metrics

d) Write short notes on code review.

e) Explain CMMI framework in detail.

Model Question Paper - II

Q.1 Attempt all (Each of 5 Marks) **(15 Marks)**

a. Choose the correct answer

1) "Are we building the right product ?". This statement best suits which of the following terms ?

a) Validation b) Verification c) CMMI d) COCOMO model

e) White box testing

2) In which of the following XP tests is the test code developed before actual task code is developed ?

a) Incremental test development from scenario
b) User involvement in test development and validation
c) Test first development
d) The use of automated test harness

3) Which step in the risk management process defines the probability and impact of each risk to determine the severity ?

a) Risk identification b) Risk analysis
c) Risk response planning d) Risk tracking and control

4) What is a critical path ?

a) Longest path in the network b) Path with the activities having float or slack time
c) Longest time to complete the project d) All of the above

5) Which of the following UML diagrams best describes the dynamic behavior of the system ?

a) Class diagram b) State chart diagram
c) Use Case diagram d) Deployment diagram

b. Fill in the blanks

a) _____ is the process of hiding the details and exposing only the essential features of a particular object.

b) CMMI is short for _____

c) The first phase of waterfall model is _____

d) Whether the system is available 99.9% of time or not is a _____ type of requirement

e) Compiler and Interpreter are _____ types of software.

c. Answer in one sentence

a) What is meant by cohesion ?

b) What does the radius of the spiral indicate in the spiral model ?

c) Define software engineering.

d) Define verification and validation.

e) State the advantage of prototype model.

Q.2 Attempt the following (Any three) **(15 Marks)**

a) State the difference between : Classic life cycle model and Prototyping model

b) What is the technical feasibility study?

c) What is agility ? Explain Test First Development model.

d) What is the role of SQA. State the tasks of SQA.

e) What is Risk Identification ? Explain the need of RMMM plan.

Q.3 Attempt the following (Any three) **(15 Marks)**

a) Define the characteristics of a software.

b) Explain the difference between PERT and CPM

c) What is pair programming? Why is it important ?

d) Explain how COCOMO - II is different from COCOMO model.

e) Calculate Cyclomatic Complexity of code which accepts a positive number from the user as input and displays whether it is an even or odd number. Find various paths and design test cases.

(15 Marks)

Q.4 Attempt the following (Any three)

a) Explain the basic principles behind project scheduling.

b) Draw a collaboration diagram for contacting a person using a mobile phone.

c) Draw an activity network for the project given below :

Activities	Duration	Precedents
A	2	-
B	3	A
C	3	-
D	2	C
E	3	D, J
F	2	E, B
G	2	F
H	4	-
J	2	H

d) State the difference between functional –oriented and object-oriented approach of system design.

e) What are the characteristics of a good SRS ?

(15 Marks)

Q.5 Attempt the following (Any three)

a) Explain requirement validation.

b) Explain aggregation and composition with suitable example.

c) State and explain the Quality metrics.

d) Write short notes on code inspection.

e) State the difference between white box testing and black box testing.

□□□

Appendix A

Solved University Question Paper of April 2018

Q. 1 (a) Multiple Choice Questions :

(i) Diagrams which are used to distribute files, libraries and tables across topology of hardware are called _____. **(1 Mark)**

(a) Deployment diagrams (b) use case diagrams

(c) sequence diagrams (d) collaboration diagrams

Ans.: (a) Deployment diagrams

(ii) The UML support event-based modelling using ____ diagrams. **(1 Mark)**

(a) Deployment (b) Collaboration

(c) State chart (d) All of the mentioned

Ans. : (c) State chart

(iii) The _____ model stipulates that the requirements be completely specified before the rest of the development can processed.

(a) Waterfall (b) Rapid Application Development (RAD)

(c) Iterative Development (d) Incremental Development

Ans. : (a) Waterfall

(iv) Project Risk factor is considered in which model ? **(1 Mark)**

(a) Spiral model (b) Waterfall model

(c) Prototyping model (d) None of the above

Ans. :(a) Spiral model

(v) Test Conditions are derived from _____. **(1 Mark)**

(a) Test Design (b) Test Cases

(b) Test Data (d) Specifications

Ans. : (d) Specifications

Q. 1 (b) Fill in the blanks :

(i) ISO stands for _____. **(1 Mark)**

Ans. : International Standards Organization.

(ii) SRS stands for _____. **(1 Mark)**

Ans. : Software Requirement Specification

(iii) SQA stands for _____. **(1 Mark)**

Ans. : Software Quality Assurance

(iv) COCOMO stands for _____. **(1 Mark)**

Ans. : Constructive Cost Model

(v) CMM stands for _____. **(1 Mark)**

Ans. : Capability Maturity Model

Q. 1 (c) Answer in 1-2 lines :

1. What is software re-engineering ? **(Chap. 1, 1 Mark)**

Ans. : [Hint : Add at the end of Section 1.4]

Reorganizing and modifying existing software systems to make them more maintainable

Re-structuring or re-writing part or all of a legacy system without changing its functionality

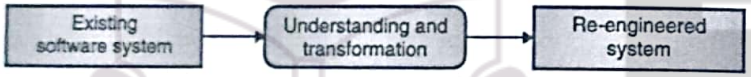


Fig. 1-Q. 1(c)(1) : Software re-engineering

2. Define uml in software engineering. *(Ans. : Refer Section 4.2.1)* **(Chap. 4, 1 Mark)**
3. What is software engineering ? *(Ans. : Refer Section 1.4)* **(Chap. 1, 1 Mark)**
4. What is software quality in software engineering ? *(Ans. : Refer Section 10.1)* **(Chap. 10, 1 Mark)**
5. What is verification and validation ? *(Ans. : Refer Section 11.1)* **(Chap. 11, 1 Mark)**

Q. 2 (a) State and explain the activities in SDLC. *(Ans. : Refer Section 2.3)* **(Chap. 2, 5 Marks)**

Q. 2 (b) Draw use case diagram for Car Rental System. **(Chap. 4, 5 Marks)**

Ans. : [Hint : Add at the end of Section 4.3]

Use case diagram for car rental system :

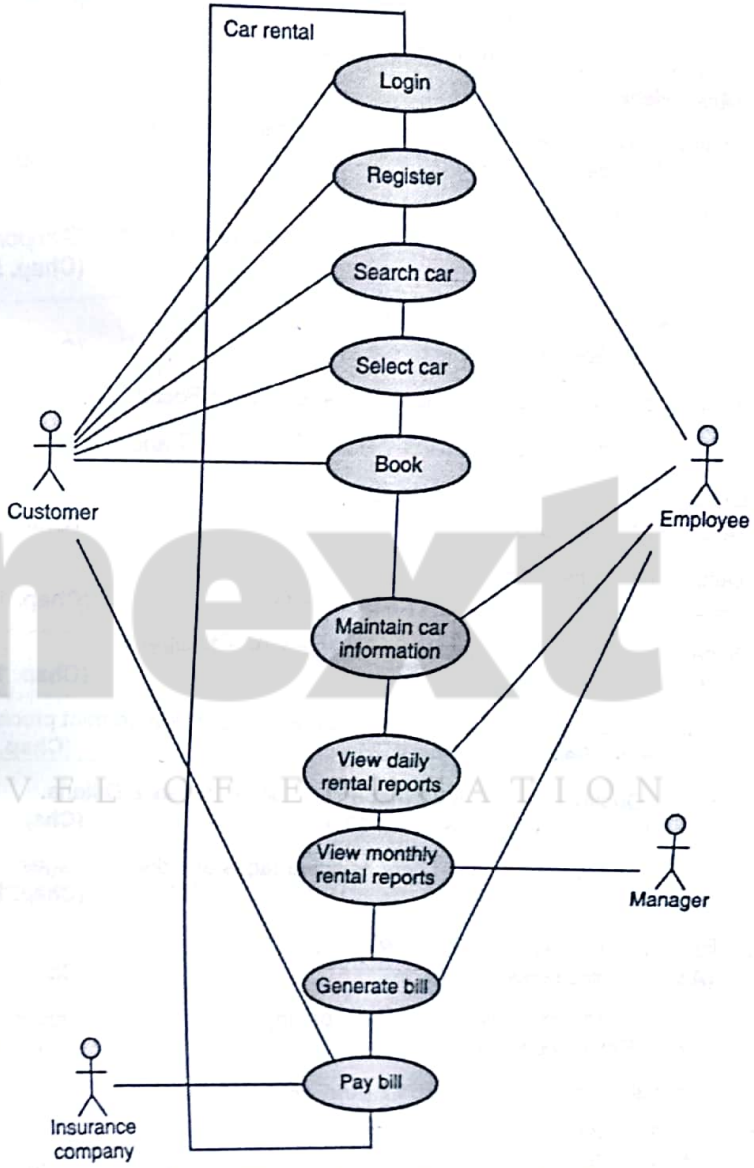


Fig. 1-Q. 2(b) : Use case diagram for car rental system

Q. 2 (c) What is SRS ? State and explain its type. *(Ans. : Refer Sections 3.5.1 and 3.5.3)* **(Chap. 3, 5 Marks)**

Q. 2 (d) What is component diagram ? Draw and explain symbols for the same. *(Ans. : Refer Section 4.10.1)* **(Chap. 4, 5 Marks)**

Q. 2 (e) Explain Agility and write its advantages and disadvantages. *(Ans. : Refer Sections 2.10 and 2.10.1)* **(Chap. 2, 5 Marks)**

Q. 2 (f) How to draw and where to use Deployment diagram ? *(Ans. : Refer Section 4.10.2)* **(Chap. 4, 5 Marks)**

Q. 3 (a) State the disadvantages of LOC. How is it different from COCOMO ? *(Ans. : Refer Sections 7.6 and 7.6.1)* **(Chap. 7, 5 Marks)**

Q. 3 (b) Explain Software user interface design. *(Ans. : Refer Sections 5.2.2(4, 5) and 5.10 (Human Interaction Component))* **(Chap. 5, 5 Marks)**

Q. 3 (c) Write the scope of software metrics. *(Ans. : Refer Sections 6.1.1 and 6.1.2)* **(Chap. 6, 5 Marks)**

Q. 3 (d) Explain software design specification. *(Ans. : Refer Section 5.7)* **(Chap. 5, 5 Marks)**

Q. 3 (e) Explain Project Scheduling. *(Ans. : Refer Sections 8.1 and 8.2)* **(Chap. 8, 5 Marks)**

Q. 3 (f) Explain Empirical Estimation model. *(Ans. : Refer Sections 7.6 and 7.6.1)* **(Chap. 7, 5 Marks)**

Q. 4 (a) Define Test Case, Test Oracle, Test Plan. *(Ans. : Refer Sections 11.12, 11.5 and 11.11)* **(Chap. 11, 5 Marks)**

Q. 4 (b) What are the errors found while doing Black Box Testing ? *(Ans. : Refer Section 11.9)* **(Chap. 11, 5 Marks)**

Q. 4 (c) What is Risk management ? Explain Software risk management process. *(Ans. : Refer Sections 9.1 and 9.3)* **(Chap. 9, 5 Marks)**

Q. 4 (d) What is Quality Assurance ? What are Quality Assurance Criteria. *(Ans. : Refer Sections 10.3.1 and 10.3.4)* **(Chap. 10, 5 Marks)**

Q. 4 (e) What is Structural testing ? Write its advantages and disadvantages. *(Ans. : Refer Section 11.8)* **(Chap. 11, 5 Marks)**

Q. 4 (f) Explain Capability Maturity Model. *(Ans. : Refer Section 10.10)* **(Chap. 10, 5 Marks)**

Q. 5 (a) State all and write down a short note on any 3 fact finding techniques. *(Ans. : Refer Section 3.8)* **(Chap. 3, 5 Marks)**

Q. 5 (b) What is coupling and cohesion ? *(Ans. : Refer Section 5.11)* **(Chap. 5, 5 Marks)**

Q. 5 (c) Explain Verification and Validation. *(Ans. : Refer Section 11.1)* **(Chap. 11, 5 Marks)**

Q. 5 (d) Define and explain ISO Quality Standards. *(Ans. : Refer Section 10.9.2)* **(Chap. 10, 5 Marks)**

Q. 5 (e) What is Cyclomatic complexity ? Explain with an example. *(Ans. : Refer Section 6.9)* **(Chap. 6, 5 Marks)**

❑❑❑

Note